Razvoj aplikacije prividne stvarnosti primjenom formalnog modela razvoja

Miloš, Karlo

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Karlo Miloš

Razvoj aplikacije prividne stvarnosti primjenom formalnog modela razvoja

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Karlo Miloš

Matični broj: 0016152280

Studij: Informacijski i poslovni sustavi

Razvoj aplikacije prividne stvarnosti primjenom formalnog modela razvoja

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Božidar Kliček

Varaždin, srpanj 2024.

Karlo Miloš

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog završnog rada jest pružiti pregled različitih modela razvoja aplikacija pomoću kojih bi se mogla izraditi aplikacija prividne stvarnosti. Prije prelaska na pregled različitih modela s njihovim prednostima i nedostacima te njihove međusobne usporedbe, pružiti će se uvid u to što je točno prividna stvarnost. Opisati će se povijesni razvoj tehnologije, gdje se prividna stvarnost sve primjenjuje, koje tehnologije omogućuju prividnu stvarnost, koja je razlika između prividne i proširene stvarnosti te na kojima se sve uređajima može iskusiti prividna stvarnost.

Nakon što su opisani različiti modeli i metodologije razvoja aplikacija, potrebno je odabrati jedan od tih modela kako bi se izradila vlastita aplikacija. Razvoj aplikacije potrebno je dokumentirati kroz sve faze razvoja, a na kraju će se pružiti detaljna analiza prednosti i nedostataka modela.

Ključne riječi: Prividna stvarnost, VR, modeli razvoja, izrada aplikacije, vodopadni model, Unity

Sadržaj

1. Uvod

Prividna stvarnost i dalje je relativno nova i nepoznata tehnologija koja tek treba iskoristiti puni potencijal. Stoga, ovome radu bit će pružen detaljan pregled prividne stvarnosti, počevši od objašnjenja što prividna stvarnost jest, a što nije, te povijesnog razvoja prividne stvarnosti. Nadalje, bit će opisane različite vrste uređaja koje omogućuju prividnu stvarnost. Prividna stvarnost se počinje sve više i više upotrebljavati u različitim industrijama i u različite svrhe, stoga će biti pružen pregled tih primjena. Prividna stvarnost predstavlja skup različitih tehnologija koje surađuju kako bi generirale virtualni svijet i omogućile korisnicima uranjanje u njega pa će biti pružen pregled različitih tehnologija koje omogućuju prividnu stvarnost.

Središnji dio ovog rada odnosi na izradu jedne jednostavne aplikacije prividne stvarnosti u nekom od formalnih modela razvoja aplikacija. Prije početka razvoja, biti će pružen uvid u raznovrsne modele i metodologije razvoja. Od formalnih modela razvoja, predstaviti će se nekoliko tradicionalnih i agilnih metoda. Konkretno, od tradicionalnih to će biti vodopadni model i njegova inačica V-Model, inkrementalni te spiralni model. Od agilnih metoda, bit će opisano ekstremno programiranje te vrlo popularni Scrum. Svaki od tih modela će biti opisan, bit će navedene prednosti i nedostaci te će biti međusobno uspoređeni.

U praktičnom dijelu rada potrebno je odabrati jedan od pruženih formalnih modela i koristiti ga za izradu aplikacije prividne stvarnosti. Model koji će se koristiti će biti vodopadni model jer će se izrađivati jednostavna aplikacija. Aplikacija će biti virtualna galerija koju će korisnici moći posjetiti kako bi vidjeli slike iz 3 različita razdoblja: renesanse, baroka i impresionizma. Cijeli razvojni proces potrebno je dokumentirati kako bi se na kraju mogle navesti dobre i loše strane modela.

2. Metode i tehnike rada

Kako bi se osigurala točnost pruženih informacija, gdje god je to moguće, koristiti će se provjereni izvori. To su pretežito knjige i znanstveni radovi i artikli koji se mogu pronaći na profesionalnoj mreži za znanstvenike *ResearchGate*.

Vlastiti grafički prikazi metoda kreirani su u online alatu Canva i pomoću Wordove ugrađene funkcionalnosti *SmartArt*.

Tijekom faze razvoja aplikacije, korišteno je nekoliko alata. Za kreiranje UML dijagrama, korišten je alat *Visual Paradigm 17.0.* Za modeliranje koristi se alat *Blender*, a za izradu aplikacije *Unity* koji koristi programski jezik C#.

3. Definicija prividne stvarnosti

Prividna ili virtualna stvarnost (eng. *Virtual Reality*, skraćeno VR) jest pojam koji definira umjetna, računalno generirana okruženja koja korisnik doživljava kao realna iskustva [1]. Autor Lowood [2] pruža sličnu definiciju prividne stvarnosti, no dodatno navodi kako je ona trodimenzionalno vizualno okruženje, a može biti i bilo koje drugo osjetilno okruženje. U takvo okruženje korisnik je potpuno uronjen (eng. *immersed*) i nema kontakta sa stvarnim svijetom. Zbog toga, stvara se iluzija da je korisnik doista „tamo“ [2].

Jaron je Lanier američki informatičar, umjetnik, autor i filozof te se smatra „ocem virtualne realnosti“ te je stvorio izraz „virtualna stvarnost“. Iako nije stvorio prvi uređaj prividne stvarnosti, zaslužan je izradu prvog modernog naglavnog VR uređaja te za komercijalizaciju prividne stvarnosti kroz prodaju VR naočala [3], [4]. Lanier je dao mnogo filozofskih definicija prividne stvarnosti od kojih se ističe sljedeća: „Prividna stvarnost jest skup uređaja koji surađuju s ljudskim osjetilnim i motoričkim organima. Uređaji omogućavaju osobi osjećaj hodanja u virtualnom svijetu, dok u fizičkom svijetu ostaje na mjestu.“ [5].

Prividna stvarnost sastoji se od 4 komponente: organizma, ciljanog ponašanja, umjetne senzorne stimulacije i svjesnosti virtualne stvarnosti. Živi organizam jest entitet na kojemu se testira prividna stvarnost. Živi organizam podrazumijeva ljude, no to mogu biti i životinje (prividna stvarnost je testirana na majmunima, voćnim mušicama, ribama,...). Ciljano ponašanje je ponašanje organizma kojeg je zamislio dizajner virtualnog iskustva. Važno svojstvo gotovo svakog iskustva prividne stvarnosti je interakcija između organizma i virtualnog svijeta. Različita iskustva sadrže različita ciljana ponašanja. Primjerice, u simulatoru za obuku pilota naglasak je stavljen na letenje, dok je u igri istraživanja prirode prisutno hodanje, planinarenje i plivanje. Tijekom boravka u virtualnom svijetu, umjetna senzorna stimulacija zamjenjuje ili nadopunjuje stvarne ulaze osjetila s virtualnim. Na primjer, osoba nosi VR naočale što dovodi do toga da osjetilo vida zaprima potpuno nove, umjetne prikaze. Umjetna stimulacija dovodi do prestanka svjesnosti stvarnog svijeta, što uzrokuje osjećaj prisutnosti u alternativnom svijetu [1].

4. Povijest prividne stvarnosti

Početak razvoja prividne stvarnosti nije točno određen. Različiti izvori navode različite osobe, koncepte i izume kao začetnike prividne stvarnosti. Zbog velike količine izvora koji tvrde da je američki informatičar Ivan Sutherland najvažniji za početak razvoja prividne stvarnosti, opis povijesti prividne stvarnosti kreće od njega.

4.1. Rani počeci prividne stvarnosti

Ivan Sutherland ističe se kao jedna od najutjecajnijih osoba u području prividne stvarnosti. 1965. godine iznosi ideju o savršenom virtualnom prikazu. Kod takvog prikaza postojala bi soba u kojoj bi računalo moglo generirati stvarne objekte. Pomoću stolice, lisica i metka Sutherland je objasnio svoju ideju. U savršenom virtualnom prikazu, na računalno generiranu stolicu moglo bi se sjesti, lisice bi mogle spriječiti kretnje rukama, a metak bi bio smrtonosan. Osim mogućnosti interakcije s takvim objektima, savršeni virtualni prikaz bi uključivao različita osjetila (njuh, sluh, okus) te vizualizaciju nestvarnih entiteta i pojava (Sutherland daje primjer trokuta sa zaobljenim rubovima i mogućnost percepcije pokreta objekta bez mase) [6].

1968. godine I. Sutherland izradio je prvi VR sustav naziva Damoklov mač (eng. *The Sword of Damocles*) i na taj način pretvorio svoj koncept u stvarnost. Damoklov mač ujedno se smatra i prvim naglavnim (eng. *Head-mounted display*, skraćeno HMD) VR uređajem. Glavne značajke uređaja bile su mogućnost praćenja pokreta glavom i stereoskopski prikaz. Slika za desno oko bila bi prikazana desnom oku, a slika lijevog oka lijevom oku. Ovaj uređaj je imao veliki značaj za razvoj uređaja prividne stvarnosti, a utjecaj je imao i na uređaje proširene stvarnosti [7].

4.2. Prividna stvarnost između 1970. godine i 2000. godine

Od 1970. godine pa sve do 2000. godine tehnologija prividne stvarnosti je znatno napredovala. GROPE je naziv prototipa sustava iz 1971. godine koji je omogućio pružanje reakcije na silu (eng. Force-feedback). 1982. godine razvijen je HMD uređaj VCASS koji je služio vojnim pilotima za simuliranje leta. NASA je također sudjelovala u razvoju VR tehnologije kroz HMD monokromatski uređaj VIVED. VPL je tvrtka koja je najzaslužnija za komercijalizaciju uređaja prividne stvarnosti. 1988. godine tvrtka VPL, koju je osnovao Jaron Lanier, stvorila je nekoliko uređaja prividne stvarnosti, od kojih se ističe Eyephone HMD. Ovim uređajem su

komercijalizirali prividnu stvarnost. 1989. godine izlazi uređaj BOOM koji se temelji na 2 CRT monitora i na mehaničkoj palici koja je mogla mjeriti poziciju i orijentaciju uređaja [7].

U razvoju tehnologije prividne stvarnosti sudjelovala su i sveučilišta pa je tako Sveučilište u Sjevernoj Karolini unaprijedilo općenitu kvalitetu HMD-a i grafike [7]. Također, Sveučilište u Georgiji zaslužno je za izradu VR simulacije rata kako bi se pomoglo vojnicima s PTSP-om [8].

CAVE je VR sustav iz 1992. godine koji koristi zidove sobe za prikaz virtualnog svijeta, čime se postiže veća kvaliteta slike, bolja rezolucija i vidno polje je veće nego kod HMD uređaja. Umjesto tipičnog HMD uređaja, koristile su se LCD shutter naočale [7].

4.3. Prividna stvarnost od 2000. godine do danas

Od 2000. do 2009. godine napredak tehnologije prividne stvarnosti je uglavnom mirovao, a interes za prividnu stvarnost je pao. Veliki skok dogodio se 2010. godine kada je na tržište stigao HMD uređaj Oculus Rift. Uređaj je donio neke novosti koje su ponovno pobudile interes za razvojem i korištenjem prividne stvarnosti. Oculus Rift prvi je imao vidno polje od 90 stupnjeva, do tada najveće vidno polje. Također, uređaj je koristio snagu računala kako bi prikazivao slike. Facebook, odnosno današnja Meta kupila je kompaniju Oculus VR i ta se kupovina smatra korakom koji je bio ključan za razvoj prividne stvarnosti jer su i druge velike tvrtke kao što su Sony, Samsung i Google odlučile razviti vlastite uređaje prividne stvarnosti [8].

Uređaji prividne stvarnosti postajali su sve više dostupni široj masi ljudi i do 2016. godine, više od 100 tvrtki je razvijalo vlastite uređaje prividne stvarnosti. 2016. godine predstavljen je uređaj HTC Vive koji je korisniku omogućio slobodno kretanje prostorijom. 2020. godine izlazi Oculus Quest 2, kasnije preimenovan u Meta Quest 2, koji je zbog svoje cijene postao jedan od najkorištenijih i najprodavanijih uređaja prividne stvarnosti na tržištu [8].

5. Prividna stvarnost nasuprot proširenoj stvarnosti

Prividna stvarnost i proširena stvarnost (eng. *Augmented Reality*, skraćeno AR) pojmovi su koje ljudi često miješaju i koriste ih kao sinonime. Iako postoje neke sličnosti, poput korištenja naglavnog uređaja i prikaza objekata koji ne postoje u stvarnom svijetu, ove dvije tehnologije se znatno razlikuju.

Kod prividne stvarnosti, korisnici su potpuno uronjeni u virtualni svijet, gdje vide samo računalno generiranu grafiku, bez ikakve interakcije sa stvarnim svijetom. Proširena stvarnost obogaćuje stvarni svijet računalno generiranim, odnosno virtualnim elementima, omogućavajući korisnicima da i dalje vide stvarni svijet uz dodatak virtualnih elemenata. Na slici 1, prikazan je virtualni lav u stvarnoj sobi. Prikaz lava, odnosno mnogih drugih životinja, znamenitosti, kemijskih i bioloških elemenata omogućava Googleova AR funkcija prilikom pretraživanja pojmova na tražilici.

Slika 1. Proširena stvarnost

Proširena stvarnost implementirana je kod brojnih mobilnih aplikacija i igara, a kao najpoznatija takva igra izdvaja se *Pokemon GO* koji je proširenu stvarnost koristio kako bi *Pokemone* i njihovo hvatanje, treniranje i borbe prikazao u stvarnom svijetu. Osim mobitela, postoje i naglavni uređaji koji omogućuju korisnicima korištenje proširene stvarnosti. Najpoznatiji uređaji su Microsoftovi HoloLens i HoloLens2 te vrlo popularni Apple Vison Pro koji je izašao 2024. godine i koji kombinira proširenu i prividnu stvarnost. Navedeni uređaji koriste se u svrhe obrazovanja, u medicini, kulturi i sportu, no zbog svoje cijene i raznih ograničenja kao što je kratko vidno polje, nemaju razinu popularnosti kakvu imaju uređaji za prividnu stvarnost.

6. Naglavni uređaji za prividnu stvarnost

Prividna stvarnost gotovo pa podrazumijeva korištenje naglavnog uređaja. Postoje iznimke kao što je sustav CAVE koji koristi 3D naočale, no naglavni su uređaji postali standard za prikaz virtualnog svijeta. Iako je razlika sve manje vidljiva, mogu se razlikovati 2 vrste naglavnih uređaja – onih koji su povezani s računalom i oni koji koriste mobilne uređaje [9].

6.1. Uređaji prividne stvarnosti povezani s računalom

VR povezan s računalom koristi snagu računala, odnosno snagu grafičke kartice i procesora kako bi obrađivao grafiku aplikacije. Računala na koja se VR uređaj veže najčešće su moćna osobna računala i Sonyjeve konzole PlayStation 4 i PlayStation 5 [9]. Spajanjem uređaja na računalo osigurava se kvalitetnija slika i bolje performanse aplikacija.

VR povezan s računalom počeo se razvijati od 2010. godine s pojavom Oculus Rifta. On je imao ugrađene zvučnike, mikrofon i brojne senzore. Koristio je Oculus Touch kontrolere, no mogao je koristiti i XBOX One kontroler. Kako bi se uređaj povezao s računalom, bio je potreban HDMI kabel. HTC Vive također je jedan od predstavnika uređaja prividne stvarnosti, a po specifikacijama je vrlo sličan Oculus Riftu, samo što omogućava korištenje SteamVR kontrolera ili bilo kojeg drugog PC kontrolera. Za razliku od Rifta i Vivea, Playstation VR (skraćeno PSVR) jest uređaj koji se spaja na konzolu Playstation 4 i Playstation 5. Uređaj koristi PlayStation Move, odnosno Dual Shock kontrolere i spaja se pomoću HDMI kabla na konzolu [10].

Meta Quest 2 i Meta Quest 3 su primjeri uređaja koji mogu biti spojeni na računalo kako bi se povećale performanse aplikacije, no mogu i samostalno pokretati aplikacije jer imaju ugrađeni procesor. Postoje 2 mogućnosti spajanja uređaja na računalo: putem kabla i putem Air Link sustava koji omogućava bežično spajanja. Na slici 2 prikazan je uređaj Meta Quest 2 koji je pomoću kabla spojen na prijenosno računalo.

Slika 2. Povezani Meta Quest 2

Najveći problem uređaja prividne stvarnosti koji se moraju spojiti na računalo je visoka cijena koju korisnici moraju platiti. Osim skupih uređaja prividne stvarnosti, potrebno je posjedovati i skupo računalo, odnosno Playstation konzolu. Korištenje slabih računala, laptopa ili MacBooka rezultira lošim performansama aplikacije. Također, težina uređaja prividne stvarnosti i manjak mobilnosti predstavljaju velike probleme prilikom korištenja uređaja [11].

6.2. Uređaji prividne stvarnosti temeljeni na mobitelu

Uređaji prividne stvarnosti temeljeni na mobitelu koriste mobitele za prikaz stereoskopske slike. Iako takav način prikaza virtualnog svijeta ne može pružiti jednako kvalitetno virtualno iskustvo kakvo mogu uređaji koji se spajaju na računalo, uređaji temeljeni na mobitelu pružaju visoku razinu fleksibilnosti i praktičnosti [12].

Google je zaslužan za stvaranje prvog takvog uređaja koji koristi mobitel za prikaz virtualnog svijeta kroz svoj Google Cardboard koji je izrađen od kartona. Iako je bio inovativan, on je bio vrlo limitiran jer je koristio procesor mobitela, nije imao mogućnost praćenja korisnikove pozicije i slično. Osim Google Cardboarda, predstavnici mobilnog VR-a su i Samsung GearVR i Oculus Go [9].

Za razliku od Google Cardboarda koji je podržavao mnogo različitih mobilnih uređaja, Samsung GearVR bio je posebno optimiziran za mali spektar Samsung uređaja, među kojima su bile različite varijante Samsunga S6 te Samsung Galaxy Note 5 [1]. Što se tiče sustava za

zvuk, niti Cardboard niti GearVR nisu imali ugrađene zvučnike i mikrofon, već su koristili zvučnike i mikrofon mobitela [10].

Oculus Go spada u uređaje novije generacije kojima više nije potreban mobilni uređaj kako bi radili. Takvi uređaji imali su integrirane zaslone i procesor i mogli su potpuno samostalno raditi [9]. Oculus Go bio je jeftin i pristupačan uređaj koji je donio brojna poboljšanja: integrirao je zvučnike, poboljšao leće, imao kvalitetniji kontroler i uspješno riješio probleme pregrijavanja uređaja, pružajući tako bolje korisničko iskustvo [12].

Važno je za napomenuti da je prekinuto ažuriranje spomenutih uređaja.

7. Primjena prividne stvarnosti

Industrija videoigra zaslužna je za početak široke primjene prividne stvarnosti. Igrači često posjeduju najnovije i najjače tehnologije, tj. tehnologije koje mogu omogućiti kvalitetno iskustvo u virtualnim svjetovima. Iako najviše cijene tehnologiju virtualne stvarnosti, igrači su vrlo zahtjevna publika. Tržište se prilagođava igračima i radi sve što može kako bi zadovoljilo njihove zahtjeve, stoga se kompanije prvenstveno fokusiraju na industriju videoigara. Tome u prilog ide činjenica da su većina aplikacija prividne stvarnosti zapravo igre [9].

Virtualna stvarnost omogućava trodimenzionalnu vizualizaciju različitih sadržaja, što se može učinkovito koristiti u obrazovanju [11]. Virtualna stvarnost se može primijeniti u obrazovanju na raznim područjima kao što geografija, povijest, biologija, kemija,... *Titans of Space* navodi se kao prva uspješna edukacijska aplikacija. Aplikacija omogućava istraživanje Sunčevog sustava i može se kupiti na distribucijskoj platformi *Steam* [9].

Prividna stvarnost pruža sigurnije i ekonomičnije alternative za različite vrste treninga, od vojnih simulacija do obuke za vožnju. Vojni treninzi često su opasni i skupi za izvesti, a uvođenjem virtualne okoline pruža se slično iskustvo bez ikakvog rizika i visokih troškova [13]. Konkretan primjer korištenja prividne stvarnosti jest obuka koju je japanski proizvođač automobila *Toyota* provela kako bi mladima ukazala na opasnosti nesmotrene vožnje [9].

Prividna stvarnost postaje sve važnija u turizmu. Bez odlaska od kuće mogu se posjetiti poznati svjetski gradovi kao što su New York i Pariz. Osim posjete lokacijama na Zemlji, neke aplikacije nude i odlazak na druge planete [9]. Osobe koje nemaju vremena ili nisu u mogućnosti putovati na jeftin i brz način mogu posjetiti razne destinacije. Naravno, virtualna putovanja imaju svoje nedostatke, kao što su gubitak prilike za susret s novim ljudima, nemogućnost isprobavanja lokalne kuhinje i doživljavanja kulture iz prve ruke. Na slici 3, prikazana je aplikacija koja omogućava posjet starom Rimu. Prema opisu [14], aplikacija omogućava posjet Rimu iz 325. godine.

Slika 3. VR Rome (Izvor: Luo Steven., 2018.)

Mnoge druge djelatnosti isto tako koriste prividnu stvarnost kako bi smanjile troškove, smanjile rizike ili povećale efikasnost rada. U arhitekturi, prividna stvarnost postaje sve važniji alat koji se koristi za modeliranje objekata. Kroz korištenje prividne stvarnosti, ideje i modeli mogu se lakše prikazati ulagačima. U medicini, prividna stvarnost koristi se u svrhu vizualizacije kako bi se poboljšala kvaliteta raznih analiza, a često se koristi i za simulaciju operacija. Psihijatri mogu iskoristiti prividnu stvarnost kako bi liječili mentalne bolesti i probleme kao što su posttraumatski stresni poremećaj (skraćeno PTSP) ili neki oblici fobija (strah od nekih vrsta životinja, strah od visine i sl.) [9].

8. Tehnologija prividne stvarnosti

Cilj prividne stvarnosti jest korisniku dati osjećaj da se stvarno nalazi u virtualnom okruženju koje izgleda toliko stvarno i uvjerljivo da korisnik zaboravi na stvarni svijet. Kako bi korisnik dobio taj osjećaj, potrebna je tehnologija koja će prevariti dijelove mozga odgovorne prvenstveno za percepciju pokreta i vida [11]. Imerzija i osjećaj prisutnosti 2 su elementa koje svako virtualno iskustvo želi postići. Imerzija predstavlja „emulaciju senzornih unosa koje organizam prima – vizualni, zvučni, haptički“. Prisutnost predstavlja „emocionalni ili intuitivni dojam prijenosa u virtualni svijet“. Odnos između imerzije i prisutnosti jest sličan odnosu između znanosti i umjetnosti gdje imerzija odgovora znanosti o virtualnoj stvarnosti, a prisutnost umjetnosti [9]. Dobro virtualno iskustvo ponudit će veću razinu imerzije, a što je veća razina imerzije, to će osjećaj prisutnosti biti veći. Kako bi se podigla razina imerzije, virtualno iskustvo mora ponuditi što kvalitetniju i prirodniju interakciju s elementima u virtualnom svijetu. Prirodnija interakcija može biti ostvarena kroz realnu simulaciju zakona fizike, integraciju prostornog zvuka i kroz mnoge druge povratne informacije koje korisnici mogu dobiti prilikom boravka u virtualnom svijetu [10].

Senzori i zaslon su ključni uređaji zbog kojih je prividna stvarnost ostvariva. Preciznije, dvije glavne tehnologije koje se temelje na senzorima i zaslonu su trodimenzionalni prikaz slike i praćenje pozicije glave [9].

Uz njih, potrebni su ulazni uređaji od kojih su najčešći posebni kontroleri za igre i platforma koja će pokretati aplikaciju koja uključuje računalni hardver, operacijski sustav, softver koji se povezuje s uređajem prividne stvarnosti,... [11].

8.1. Stereoskopski trodimenzionalni prikaz slike

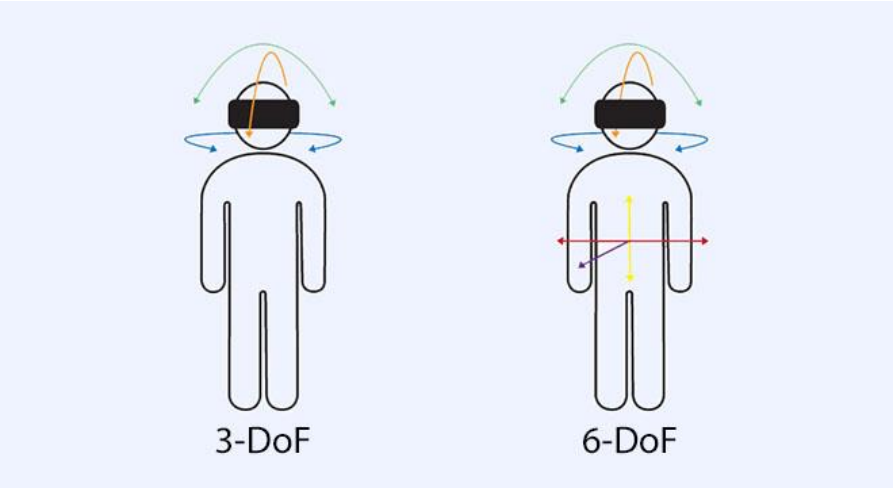
Stereoskopski prikaz slike prilično je stara tehnologija koja potječe iz 19. stoljeća, točnije stvorena je 1838. godine od strane Charlesa Wheatstonea [1]. Glavno obilježje prikaza jest stvaranje efekta paralaksa koji može prevariti mozak na način da stvori dojam trodimenzionalnosti objekata. Kako bi se efekt stvorio, potrebno je prikazati jednu sliku jednom, a drugu sliku drugom oku [9]. Osim više slika, koristi se realistična optička distorzija i leće koje mogu stvoriti stereoskopsku sliku [11]. Leće su zadužene za produkciju stereoskopske slike i one pokrivaju toliko široki kut da dolazi do distorzije slike, stoga je potrebno raditi korekciju distorzije. Korekciju radi grafički softver SDK koji je dužan stvoriti inverziju distorzije. Kao rezultat, dobije se vidno polje koje pokriva dovoljno veliki kut. Osim korekcije distorzije, izvodi

se i korekcija kromatske aberacije kako bi se slika izoštrila. *Screen-door* efekt također može biti problem kod stereoskopskog prikaza slike, a pojavljuje se zbog niske rezolucije pri kojoj korisnici mogu vidjeti piksele. Kako bi se umanjio taj problem, razvijene su tehnologije zamućivanja piksela koja se zove *Pixel smearing* i *foveated renderinga* – tehnologija kojom se prikazuju detalji veće rezolucije na mjestima gdje oči gledaju [9].

Kako bi se *foveated rendering* mogao raditi, potreban je sustav za praćenje očiju. Sustav se može implementirati na 3 načina. Prvi način jest elektrookulografija (skraćeno EOG) koja prati oči na temelju elektroda postavljenih oko očiju. Druga metoda koristi kontaktne leće s magnetskom zavojnicom koja omogućava praćenje na temelju promjena u elektromagnetskom polju. Treća metoda jest videookulografija (skraćeno VOG). Ona koristi infracrvene zrake kojima obasjava oko i na temelju refleksije prati poziciju oka. Kontaktne leće osiguravaju najveću točnost u mjerenju pozicije oka, ali zbog niskih troškova i iznimno niske razine invazivnosti, u praksi je se najčešće koristi VOG metoda [1].

8.2. Praćenje pozicije

Praćenje pozicije glave druga je najvažnija tehnologija koja omogućava korištenje prividne stvarnosti. Stereoskopski trodimenzionalni prikaz slike stvara dojam trodimenzionalnosti slike, dok praćenje pozicije glave omogućava ažuriranje slike prema promjenama u položaju glave. Za praćenje pozicije glave koristi se IMU, odnosno *Inertial Measuring Unit* [9]. IMU jest elektronički senzor koji koristi akcelerometar kako bi mjerio i pratio ubrzanje pokreta, žiroskop kako bi se pratila orijentacija glave te magnetometar za mjerenje gravitacijskih sila. IMU kod uređaja prividne stvarnosti omogućava 3 stupnja slobode (eng. *Degrees of freedom*, skraćeno DOF) od tipično njih 6 – rotiranje oko *x*, *y*, i *z* osi (*eng. pitch, roll, yaw)* [15]. Druge 3 razine predstavljaju pozicioniranje prema *x*, *y* i *z* koordinatama i prema tome mogu pratiti poziciju u prostoru korisnika [7]. Primjer uređaja koji ima 3DOF jest Oculus Go [12], a 6DOF uređaj je, na primjer, Oculus Quest 2. Na slici 4, vizualno je prikazana razlika između 3DOF i 6DOF.



Slika 4. Razine slobode (Izvor: Barnard Dom, 2023.)

8.3. Praćenje pozicije u prostoru

Kao što je spomenuto u poglavlju 8.3., IMU omogućava samo tri stupnja slobode. Druga tri stupnja koja se odnose na praćenje pozicije u prostoru moraju se ostvariti kroz neke druge tehnologije [1]. Dvije popularne metode za praćenje pozicije su metoda „unutra prema van“ (eng. inside-out) i „izvana prema unutra“ (eng. outside-in).

Metoda inside-out koristi senzore koji se nalaze na uređaju čiju je poziciju potrebno pratiti, a to je kod prividne stvarnosti naglavni uređaj. Pozicija se kod takvog pristupa mjeri analizom promjene pozicije uređaja u odnosu na okoliš, odnosno na značajke u okolišu. Za praćenje značajki mogu biti korišteni markeri, a postoji mogućnost praćenja i bez njih. Markeri su oznake koje je lako detektirati i nalaze se na specificiranim mjestima. QR kodovi su primjer markera koji služe kao referentne točke kako bi se pratila pozicija. Ako se ne koriste markeri, tada sustav za praćenje pozicije u prostoru gleda stvarne oblike koji postoje u prostoru i pozicionira se prema njima. HTC Vive i Oculus Quest su primjeri uređaja koji koriste inside-out metodu [16].

Metoda outside-in predstavlja inverziju metode inside-out. Kod outside-in metode, senzori su postavljeni na stacionarne točke iz kojih mogu snimati pokrete uređaja kojemu prate poziciju. U tom slučaju, markeri, koji su najčešće infracrveni, nalaze se na uređaju, a vanjski senzori ih prate. Primjer outside-in sustava jest PlayStation VR [17].

9. Formalni modeli razvoja aplikacija

Izraditi kvalitetnu, održivu i stabilnu aplikaciju nije jednostavan zadatak. To je kompleksan proces kojega rade iskusni inženjeri, no kako tehnologija postaje dostupna sve većem broju ljudi i kako raste interes za izgradnju aplikacija prividne stvarnosti, osobe koje nisu eksperti u ovome polju također mogu sudjelovati u razvoju aplikacija prividne stvarnosti [18].

U usporedbi s razvojem aplikacija za stolna računala i web, razvoj aplikacija prividne stvarnosti je puno kompliciraniji. Tehnologija koja služi za izradu virtualnih iskustava nije toliko stara kao što su tehnologije pomoću kojih se razvijaju web i desktop aplikacije, a to znači da sa sobom donosi i puno teže probleme koji se trebaju riješiti tijekom razvoja. Za izradu jednostavne web aplikacije, dovoljno je znati kako označiti dijelove teksta. Za izradu jednostavne aplikacije prividne stvarnosti, potrebno je puno više vremena i znanja jer se moraju znati podesiti geometrijske transformacije, mora se postaviti scena, potrebno je pravilno uključiti sve skripte koje omogućavaju kretanje korisnika [18].

Za izradu tipične jednostavne web aplikacije, dovoljna su 2 tipa programera: onaj koji zna stvoriti dobro korisničko iskustvo (frontend programer) i onaj koji je zadužen za server i bazu podataka (backend programer). Tako jednostavna podjela nije prisutna kod izrade aplikacija prividne stvarnosti. Za nju su potrebni vješti animatori, osobe koje znaju modelirati trodimenzionalne objekte u aplikacijama kao što su Blender ili Autodesku Maya. Zatim su potrebni i grafički dizajneri, pisci ako se radi o igri koja ima dijaloge, priču i svijet igre, tehničari za zvuk i naravno programeri. Oni svi moraju dobro surađivati kako bi stvorili kvalitetnu aplikaciju jer ako neka od komponenti aplikacije nije na zadovoljavajućoj razini, korisnici i recenzenti neće biti zadovoljno čime će se smanjiti prodaja, a time i zarada od aplikaciji [18].

Tipične aplikacije koriste već ustaljene, poznate i podržane tehnologije kao što su tipkovnica i miš. Prividna stvarnost ima vlastita rješenja koja nisu tipična, standardizirana, podržana od operacijskog sustava, nemaju veliko tržište pa je važna stvar prije kretanja na izradu aplikacije prividne stvarnosti odrediti ispravni hardver i integrirati ga u sustav. Naravno, za to je potrebno i iskustvo i znanje osoblja zaduženog za izradu aplikacije prividne stvarnosti [18].

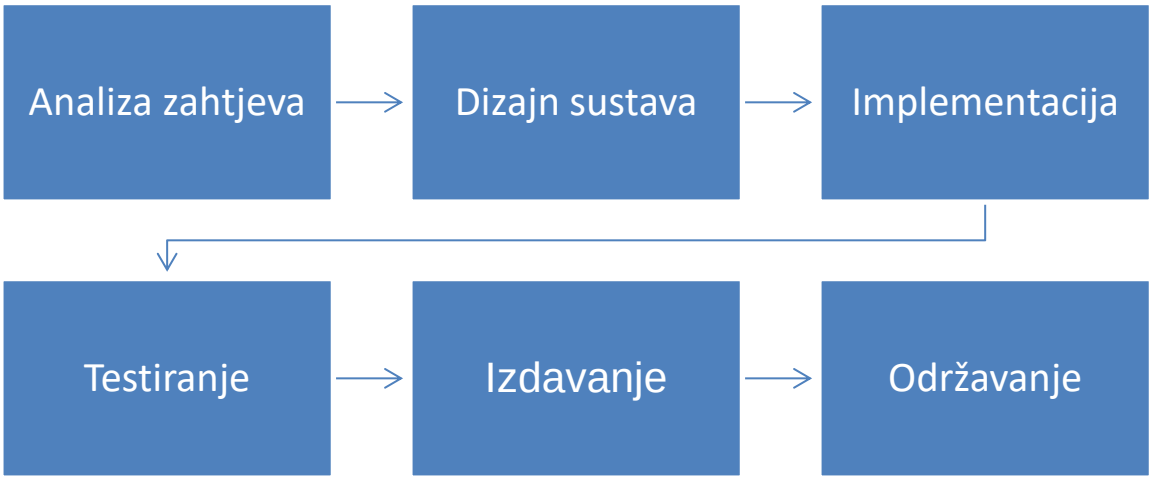
Kako bi se riješili problemi kod razvoja različitih vrsta aplikacija, pa tako i aplikacija prividne stvarnosti, pruženi su razni modeli i metodologije razvoja aplikacija. Oni služe kao

putokazi kako izraditi kvalitetnu aplikaciju, a mogu ih koristiti i eksperti i osobe koje to nisu. Neki od najpoznatijih modela razvoja su vodopadni, inkrementalni, spiralni model i agilni pristup. Ovi modeli mogu se koristiti za izradu bilo kakvih aplikacija, stoga su razvijene i posebne metodologije koje se fokusiraju samo na prividnu stvarnost. Jedne od najpoznatijih takvih metodologija su TRES-D, metodologija za razvoj aplikacija prividne stvarnosti VR trening prema Andrzeju Paszkiewiczu et al., metodologija za stvaranje afektivnog VR iskustva prema Nicoli Doziu et al. i metoda temeljena na Unityju 3D.

9.1. Vodopadni model razvoja aplikacije

Vodopadni model, drugim imenom linearni sekvencijalni model životnog ciklusa (eng. *Linear sequential life cycle model)*, najstariji je model razvoja aplikacije u kojemu se svaka faza razvoja odvija i završava prije iduće faze. Artefakti koji su dobiveni u jednoj fazi služe kao ulaz u drugu fazu [19]. Zbog svoje jednostavnosti, a i broja uspješnih projekata koji su uspjeli koristeći ovu metodu, vodopadni se model često koristi za izradu softvera [20].

Model je podijeljen na 6 faza razvoja koje su vidljive na grafici ispod.



Slika 5. Vodopadni model (prema Ali Khanu, 2023.)

Prva faza u modelu je analiza i specifikacija korisničkih zahtjeva. Tijekom ove faze, potrebno je precizno utvrditi funkcionalne i nefunkcionalne zahtjeve sustava koje korisnici očekuju da im se isporuče. Nakon što se odrede, oni se dokumentiraju i pohranjuju u repozitorij. Zahtjevi su zapisani na visokoj razini apstrakcije, a njihov broj ovisi o količini resursa s kojima upravlja organizacija zadužena za njihovu implementaciju. Na kraju ove faze, potrebno je provesti analizu kvalitete zahtjeva; razumiju li svi zaposlenici sve zahtjeve,

prihvaćaju li svi sve zahtjeve i zatim se provodi provjera jesu li svi zahtjevi identificirani te ako postoji još koji nedokumentirani zahtjev, on se zapisuje u repozitorij [21].

Prikupljeni podaci, tj. zahtjevi iz prethodne faze koriste se kako bi se ispravno dizajnirao sustav [20]. Tijekom dizajna sustava, zapravo se dizajnira arhitektura sustava [21]. Konkretno, dizajniraju se algoritmi, baza podatka, stvaraju se logički dijagrami, konceptualni dizajni, UI dizajn i slično [19].

Na temelju stvorenog dizajna sustava, započinje implementacija softvera. Ovo je faza u kojoj se kodira rješenje, tj. dizajn sustava koji je nastao na temelju zadanih funkcionalnosti se „prevodi“ u kod [20].

Verifikacija i validacija su tipični procesi koji se odvijaju u fazi testiranja. Verifikacijom se utvrđuje radimo li softver kako treba, a validacijom radimo li pravi softver. Time se provjerava prati li softver zahtjeve koje je korisnik zadao. Cilj testiranja je pronaći te otkloniti greške sustava [19]. U fazi testiranja provodi se i testiranje kvalitete, a ne samo njegovih funkcionalnosti. Na primjer, mjere se performanse sustava na različitim hardverima [21].

Nakon što je softver testiran i sve je u redu, kreće faza izdavanja. U njoj je potrebno pripremiti svu dokumentaciju potrebnu za izdavanje, a to su na primjer upute za instalaciju, priručnik kako se aplikacija koristi i slično [21].

Faza održavanja obuhvaća sve aktivnosti kojima se modificira softver tako da ostane prihvatljiv i korisnicima, a i nekim vanjskim faktorima, na primjer zakonima ili trendovima na tržištu [20]. Također obuhvaća i popravljanje greški, poboljšavanje kvalitativnih obilježja softvera kao što su performanse, kompatibilnost, pristupačnost i tako dalje [19].

Vodopadni model je strogo definiran i jednostavan model s jasnim prijelazima faza. Dobro ga je koristiti kod manjih projekata koji se ne mijenjaju i za koje su funkcionalni zahtjevi potpuno jasni. Također, dobra stvar u vezi vodopadnog modela jest dokumentiranost – svi zadaci, procesi i rezultati tijekom izrade softvera su jasno definirani i dokumentirani [19].

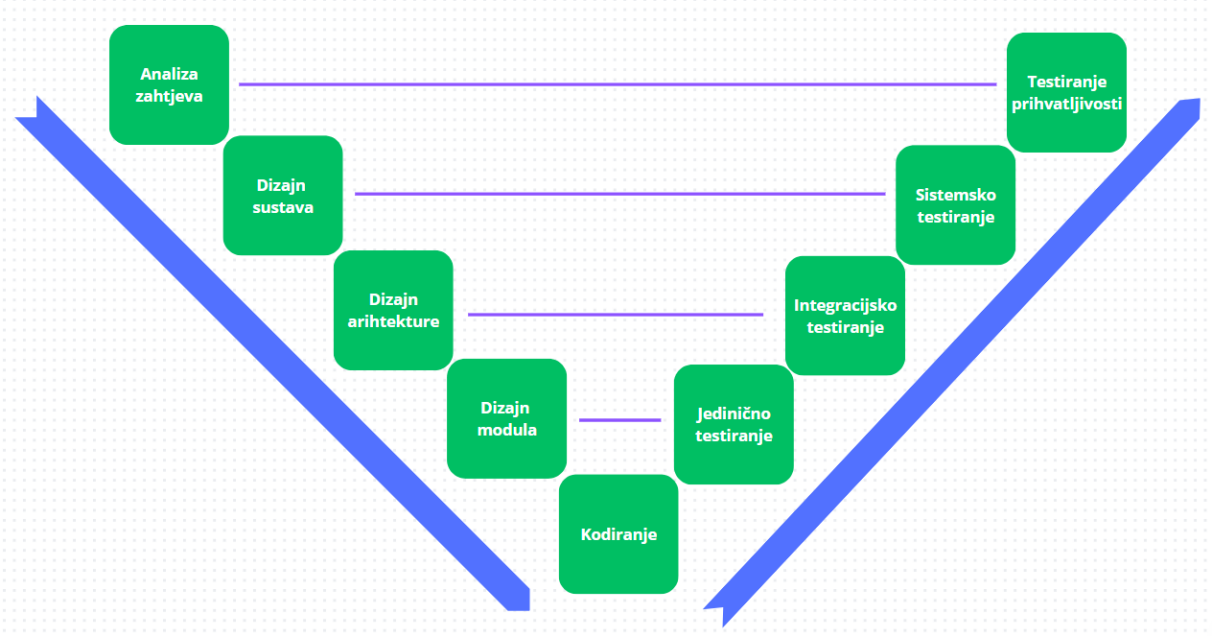
Vodopadni model je ekstremno rigidan zbog načina na koji se izvršavaju faze. To dovodi do slučaja da će softver koji radi nastati jako kasno te će time i dolazak na tržište (eng. Time-To-Market) doći jako kasno. Ako se radi o većem, kompleksnom projektu, faza kodiranja će biti zahtjevna i opširna, a integracija rješenja će se dogoditi na kraju. Naravno, to je uvijek riskantno jer ako se otkriju bilo kakvi problemi, trebati će puno vremena i novca da bi se oni

riješili. Tu se javlja i problem prekasnog testiranja. Ako se tijekom testiranja utvrdi da je neka funkcionalnost pogrešno implementirana, bit će ju teško ispraviti jer je sve već gotovo.

Pošto ovaj model nije inkrementalni, tj. svaka faza se izvodi jednom, ovaj model nije prikladan za projekte koji će zahtijevati konstantna ažuriranja i popravke. Najbolje rečeno, ovo je stabilan model za stabilne projekte za koje postoji jako mali rizik da će se nešto mijenjati [20].

9.1.1. V-Model kao proširenje vodopadnog modela

V-Model predstavlja modificiranu verziju klasičnog vodopadnog modela. Kao i kod vodopadnog modela, faze se izvode sekvencijalno, no s dodatkom da se jasno naglašava povezanost faze testiranja s fazom razvoja [22]. Kao što se može vidjeti na slici 6**,** nakon analize zahtjeva slijede 3 aktivnosti dizajna; dizajn sustava, arhitekture i modula. Sa svakom fazom se planira testiranje pa tako tijekom analize zahtjeva, planira se testiranje prihvatljivosti, tijekom dizajna sustava, planira se sistemsko testiranje, za dizajn arhitekture integracijsko testiranje, a tijekom dizajniranja modula, planiraju se jedinični testovi [22].



Slika 6. V-Model (prema S. Atanasijeviću, 2022.)

Tijekom analize zahtjeva, planira se i testiranje prihvatljivosti jer testovi prihvatljivosti koriste zahtjeve koje klijent traži. Dizajn softvera je podijeljen na 3 faze. Prva faza, dizajn sustava, definira hardver i elemente za komunikaciju. Tijekom ove faze planira se i piše plan

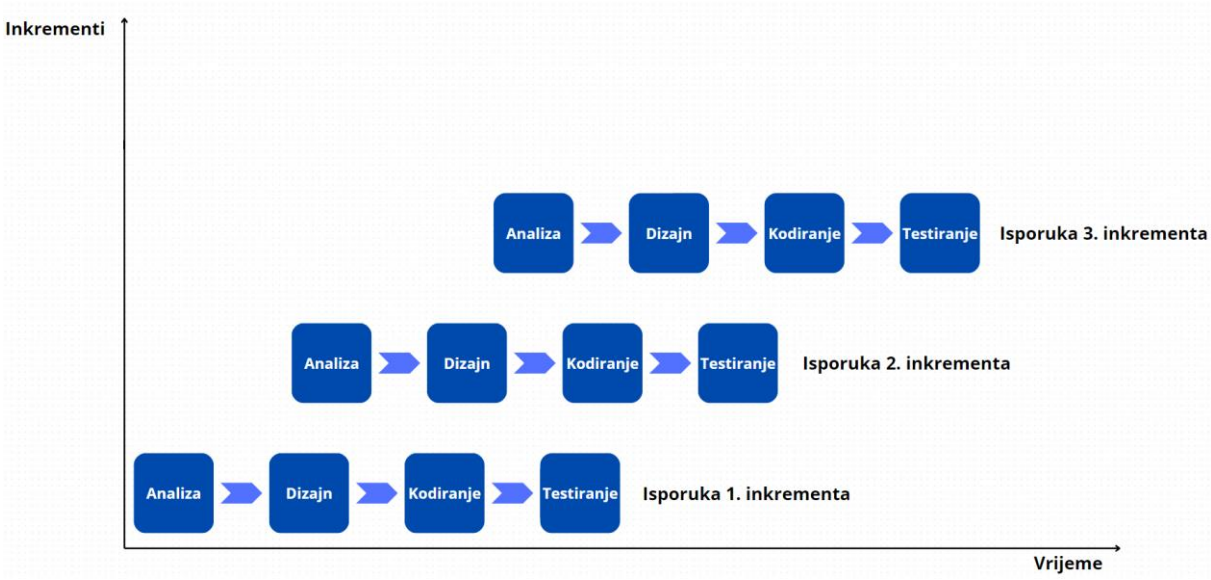
sustavskog testiranja [22]. Dizajn arhitekture podrazumijeva donošenje odluke o tehničkom pristupu. Dizajn sustava se dijeli na manje komponente, tj. na module. Svaki modul ima svoju funkcionalnost. Tijekom faze određuje se komunikacija između modula te komunikacija s drugim sustavima [23]. Tijekom dizajna arhitekture, dizajniraju se i dokumentiraju testovi integracije. Dizajn modula pruža uvid u unutrašnjost modula aplikacije (niža razina od arhitekture). Tijekom ove faze, cilj je uspostaviti kompatibilnost modula u arhitekturi, ali važan dio jest i dizajniranje jediničnih testova koji su zaduženi za testiranje unutrašnjosti modula. Sve ove aktivnosti su dio lijeve strane V-Modela, odnosno to se naziva fazom verifikacije V-Modela. [22].

Odmah nakon kodiranja, kreće faza validacije V-modela. Prvi su na redu jedinični testovi kojima se u ranoj fazi eliminiraju greške kod modula softvera. Testiranje integracije odnosi se na testiranje komunikacije između modula, a povezano je s arhitekturom sustava. Testiranje sustava provjerava sustav kao cjelinu i njegovu komunikaciju s drugim sustavima. Ovo testiranje je opširnije i od jediničnih i od integracijskih testova. Niti jedan od ovih testiranja ne testira proizvod u stvarnom okruženju, već su za to dizajnirani testovi prihvatljivosti. Zaduženi su za otkrivanje nekompatibilnosti s drugim sustavima te za otkrivanje nefunkcionalnih problema kao što su performanse i skalabilnost [22].

V-Model ima uglavnom iste prednosti i nedostatke koje ima i model vodopada. I dalje je to rigidan model koji nije sklon promjenama i nije namjenjen kompleksnim i velikim projektima, no u odnosu na model vodopada, postoji veća šansa da će projekt biti uspješan. Razlog tome jest što se veliki naglasak stavlja na testiranje odmah od početka životnog ciklusa [24].

9.2. Inkrementalni model razvoja aplikacije

Inkrementalni model razvoja softvera izgledom podsjeća na vodopadni model (slika 7), no, za razliku od samo jednog ciklusa prisutnog u vodopadnom modelu, inkrementalni model uključuje više razvojnih ciklusa. Inkrementalni model izgleda kao više vodopadnih modela jer se zahtjevi dijele u podskupove i onda se prema njima stvaraju ciklusi razvoja. Svaki ciklus se može dodatno podijeliti do te mjere da svaki modul u aplikaciji ima vlastiti ciklus razvoja – od analize zahtjeva, preko dizajna i implementacije pa do testiranja i izdavanja [25]. Inkrementalni model je primjer evolutivnog modela jer se softver s vremenom poboljšava i postaje sve bolji; on evoluira tijekom procesa izrade [26].



Slika 7. Inkrementalni model (prema M. Shah, 2016.)

Nakon prvog inkrementa, kada se isporuči prvi modul, dobiva se radna verzija softvera, a svaka sljedeća iteracija nadograditi će tu radnu verziju. Stoga, ovaj model dobro je koristiti ako je potrebno što brže isporučiti prvu verziju softvera. Također, korisnički zahtjevi bi trebali biti jasni, a oni najbitniji zahtjevi se ne bi smjeli mijenjati. Neki manji detalji oko zahtjeva se mogu mijenjati jer se vrlo lako mogu implementirati u nekoj iteraciji [25].

Najveća prednost ovog modela jest komunikacija s klijentom, koju, na primjer, vodopadni model nema. Kod vodopadnog modela nema prve radne verzije, već se softver isporučuje u cijelosti. Ako se koristi inkrementalni model, svaki inkrement će rezultirati radnom verzijom softvera, a klijent će moći dati povratne informacije o tome što je dobro, a što nije. Također, još jedna prednost jest i fleksibilnost modela jer se mogu brže i jeftinije implementirati promjene. Isto vrijedi i za funkcionalnosti – ako klijent želi neku novu funkcionalnost, ona se može dodati i biti će dostupna u novoj radnoj verziji, odnosno u novom prototipu aplikacije [25].

Iako donosi mnoge prednosti, inkrementalni model nije savršeni model. U usporedbi s vodopadnim modelom, troškovi su veći. Potrebno je provesti opsežno planiranje i dizajn sustava kako bi se maksimalno smanjile greške u sustavu jer se one kasnije jako teško popravljaju, a ako se stalno nešto mijenja, cijeli model se može svesti na kodiranje pa odmah popravljanje [25].

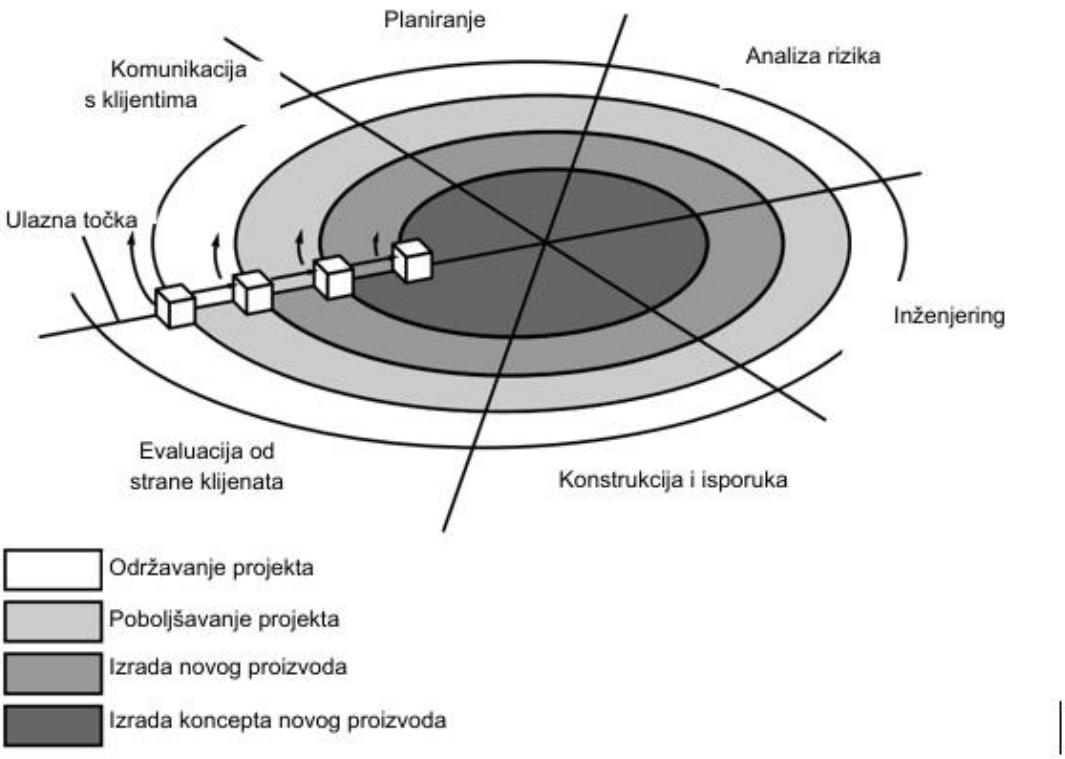
9.3. Spiralni model razvoja aplikacije

Spiralni model, iako prilično star, vrlo se često koristi i danas. On kombinira najbolje stvari od nekoliko modela, kao što su vodopadni, iterativni i inkrementalni model te pokušava riješiti njihove probleme. Velika razlika između spomenutih modela i spiralnog modela jest što spiralni model pridaje veliku važnost analizi rizika [22]. Kako je model evolutivni, on omogućava brzo izdavanje malih inkrementalnih isporuka. Rezultat početnog inkrementa nije gotovo rješenje, već to može biti prototip, model zapisan na papiru i slično. Kasniji inkrementi donose sve kompletnije verzije sustava [26].

Razvojni proces podijeljen je na *područja zadataka,* a broj područja varira od 3 do 6, zavisi o kojoj alternativi modela se radi. Pressman [26] u svojoj knjizi navodi 6 područja: komunikacija s kupcem, planiranje, analiza rizika, inženjering, konstrukcija i izdavanje te evaluacija od strane korisnika. Autori Ali Munassar N. i Govardhan A. [24] opisuju 4 područja: postavljanje ciljeva, procjena rizika, razvoj i validacija te planiranje.

Svako područje sadrži svoje zadatke. Oni su također varijabilni te njihova količina i formalnost ovise o projektu. Ako se radi o manjem projektu, zadataka će biti nešto manje i oni vjerojatno neće biti toliko formalni kao kod velikog projekata gdje zadataka ima više [26].

Kao što je vidljivo na slici 8, razvojni proces započinje u sredini i kreće se u smjeru kazaljke na satu prema van. Sa svakom iteracijom evoluira projekt. Prvih nekoliko iteracija rezultira softverskim konceptima, zatim nastaje prototip koji se nadograđuje i proces se ponavlja sve dok se ne dođe do trenutka kada se dobije sofisticirana verzija softvera koja se svakom iteracijom nadograđuje. Na kraju, svakom se iteracijom održava softver [26].



Slika 8. Spiralni model (prema R. Pressmanu, 2000.)

Na slici 8. prikazan je spiralni model prema R. Pressmanu. Faza komunikacije s kupcem obuhvaća sve zadatke s kojima se pribavljaju zahtjevi korisnika. Planiranje obuhvaća zadatke definiranja resursa, rokova, ključnih ciljeva, procijenjenih troškova, postavljanje kontrolnih točaka i svih drugih informacija vezanih za projekt [26]. Analizom rizika, identificiraju se, procjenjuju i prate tehnički i upravljački rizici, odnosno procjenjuje se postoji li realna mogućnost (tehničke) izvodljivosti u okvirima budžeta [22]. Također, kako bi se smanjili rizici, pružaju se alternativna rješenja na razmatranje [24]. U fazi inženjerstva, izvode se zadaci kojima se prikazuje reprezentacija sustava. Nakon toga, slijedi konstrukcija rješenja i izdavanje prototipa ili softvera. Ova faza uključuje i testiranje, pisanje korisničke dokumentacije i trening klijenata. Posljednja faza u Presmannovom spiralnom modelu jest klijentska evaluacija rješenja. U ovoj fazi potrebno je prikupiti povratne informacije kako bi se popravile loše i poboljšale nove funkcionalnosti [26].

Kako bi ovaj model bio najefektivniji, trebao bi se upotrebljavati kod velikih i dugoročnih projekata koji su skloniji promjenama. Pošto se ovaj model fokusira na analizu rizika, svoj puni potencijal ostvaruje kod projekata visokog rizika gdje je sama procjena ključna te gdje je održavanje troškova u granicama budžeta ključno. Model također osigurava veliku razinu

fleksibilnosti jer se može nositi s naknadnim promjenama i novim, složenim zahtjevima. Uz sve to, poboljšana je i komunikacija s klijentima. Oni mogu dobiti prototip proizvoda vrlo rano i mogu evaluirati proizvod pružajući time korisne povratne informacije [22].

Naravno, kao i svaki model, spiralni model također ima svoje nedostatke. Prvi i najočigledniji problem jest taj što previše ovisi o analizi rizika. Cijela uspješnost softvera temelji se na tome koliko je analiza rizika uspješno provedena [24]. Ključna je stručnost oko procjene rizika jer ako se veći rizik ne otkrije na vrijeme, problemi će se zasigurno pojaviti, a uspješnost projekta će automatski biti ugrožena. Za razliku od već opisanog vodopadnog i inkrementalnog modela, ovaj model nije dobar za primjenu kod manjih projekata, što je i logično jer će rizici biti manji i neće biti potrebe za mnogim iteracijama, a ako se već koristi kod manjih projekata, može postati jako skup [26].

9.3.1. WinWin model

Jedan od poznatih modela koji se temelji na spiralnom modelu jest WinWin model. On je nastao zbog mogućih nesuglasica u komunikaciji s klijentima. Tijekom pregovora, gotovo uvijek će netko biti nezadovoljan te će se morati raditi kompromisi, bili oni tehnološki, u vidu performansi ili budžeta. Stoga, pomoću WinWin modela, nastoji se doći do rješenja s kojim će sve strane biti zadovoljne. Na kraju bi klijenti trebali biti zadovoljni jer će dobiti kvalitetan i funkcionirajući softver, a softverski će inženjeri moći raditi s realnim budžetom i rokovima. Ključni koraci ovog modela su identifikacija zainteresiranih strana, tj. klijenata i njihovih uvjeta za koje žele da se ispune. Glavni korak nakon identifikacije njihovih želja jest uskladiti te želje kako bi klijenti na kraju bili zadovoljni sa aplikacijom, a da pri tome softverski inženjeri mogu raditi s realnim budžetom i vremenskim rokovima [26].

10. Agilni pristup razvoju softvera

Agilni pristup razvoju softvera temelji se na sprintovima, poslovima koji se brzo provode, a isporučuju se u kratkim ciklusima. Cilj ovakvog pristupa jest smanjiti troškove projekta. Fokus agilnog pristupa jest raspodjela zadatka na manje dijelove. Svaki dio ima svoja obilježja, ciljeve i zadatke. Zbog raspodjele, upravljanje projektom i njegovo održavanje je lakše te se na jednostavan način može pratiti napredak projekta [27].

Bitan aspekt agilnog pristupa su sastanci nakon svakog ciklusa. Obično se sastaju zaposlenici, menadžeri i klijenti i dioničari. Na sastanku se predstavlja napredak projekta i dobiva se povratna informacija svih uključenih dionika pa tako, na primjer, dioničari se mogu žaliti na brzinu izrade softvera, a tim programera može predstaviti nove ideje/probleme za koje menadžeri možda nisu bili svjesni. Takva komunikacija je ključna za uspjeh projekta pa je zato agilni pristup često korišten. Kroz komunikaciju, svi dionici su konstantno uključeni u razvoj, svi su stalno jednako upućeni u projekt i svi mogu ukazati na probleme koji se zatim mogu bezbolno riješiti, uz minimalno utrošeno vrijeme i troškove [27].

U svojoj knjizi [27], G. Caldwell navodi 3 stvari koje čine agilni pristup razvoju softvera različitim od drugih metoda. Prva stvar je da je agilni pristup segmentiran prema dizajnu, što znači da je cijeli projekt podijeljen na nekoliko iteracija, a svaka iteracija traje oko 4 tjedna. Druga stvar jest što se agilni pristup temelji na vremenu. Svaki segment aplikacije dio je neke iteracije i vremenski je zacrtano do kada bi on trebao biti gotov. Na taj se način potiče veća produktivnost, što rezultira bržim vidljivim rezultatima. Treća stvar koja razlikuje agilni pristup od drugih metoda jest takva jednostavnost da svi dionici uključeni u razvoj softvera mogu razumjeti napredak, ciljeve i zadatke. To je posebno bitno jer se agilni pristup temelji na komunikaciji i objašnjavanju.

Osim razlika, G. Caldwell u svojoj knjizi [27] navodi i 12 jednostavnih principa koji vode agilni pristup. Prvi princip govori da je najveći prioritet zadovoljiti klijente kroz ranu, brzu i kontinuiranu isporuku softvera. Drugi princip govori kako ne smije postojati strah od naknadnih promjena, već se te promjene koriste kako bi se postigla konkurentska prednost klijenata. Sljedeći princip ukazuje na to da bi se isporuka trebala događati često, u roku od nekoliko tjedana do nekoliko mjeseci. Sljedeća 2 principa odnose se na zaposlenike. Koordiniranje zaposlenika se treba odvijati svakodnevno te bi se projekti trebali graditi samo oko motivirane grupe ljudi kojima je pruženo sve što im treba – od okruženja do povjerenja. Šesti princip govori o važnosti komunikacije s naglaskom da bi ta komunikacija trebala biti licem u lice.

Sedmi princip odnosi se na to kako mjeriti uspješnost tima. Ako softver na kraju iteracije radi, znači da projekt napreduje kako bi i trebao. Stoga, treba se poticati da u svakoj iteraciji dodaju neku novu funkcionalnost, ali treba biti oprezan da se ne požuruje previše jer bi to moglo rezultirati smanjenjem produktivnosti. Osmi princip govori o kontinuiranom radu bez perioda neaktivnosti, što je mana većine drugih modela. Deveti princip naglašava važnost kontinuirane pažnje na tehničku izvrsnost i dobar dizajn za poboljšanje agilnosti kako bi se stvorio kvalitetan proizvod za sve dionike. Deseti princip govori koliko je jednostavnost važna. Ona je srž agilnog pristupa jer ako nešto nije jednostavno, potrebno je puno vremena da se to objasni, a čim se nešto mora dugo objašnjavati, gubi se puno vremena. Pretposljednji princip odnosi se na način na koji se dolazi do najbolje arhitekture i dizajna softvera, a to je preko samo-organizirajućih timova, a ne preko menadžera koji donosi sve odluke. Posljednji, dvanaesti princip, govori kako tijekom intervala, tim razmatra kako postati efektivniji pa prema tome usklađuje svoje ponašanje. Na taj način može se brzo prilagoditi novim zahtjevima ili uvjetima. To se može postići redovitim i čestim sastancima na kojima se identificiraju problemi u učinkovitosti tima, nakon čega se donose odluke o njihovom rješavanju i poboljšanju.

Kroz ove principe, već se primjećuju prednosti modela. Komunikacija u timu i s ostalim dionicima, jednostavnost modela, mogućnost izmjena zahtjeva tijekom procesa izrade softvera, brze i kvalitetne isporuke i kontinuirano poboljšavanje softvera izdvajaju se kao najveće prednosti modela [22].

Model sa sobom donosi i neke nedostatke. Cijeli tim mora biti posvećen svojemu zadatku i svi moraju surađivati tijekom cijelog procesa razvoja i tek je tada model najuspješniji. Također, timovi su mali i svaki član tima mora znati više-manje sve detalje o projektu. Kod tradicionalnijih metoda, planiranje se provodi kvalitetno, a kod agilnog pristupa može doći do nekih manjkavosti jer je teško odrediti rok isporuke (a cijeli pristup se temelji na kontinuiranim isporukama) pa neke funkcionalnosti možda neće biti implementirane na vrijeme. Manjak dokumentacije je još jedan problem jer se agilni pristup ne fokusira na dokumentaciju i često ju zaposlenici ne pišu, a to može rezultirati manje kvalitetnim i slabije održivim softverom [22].

Na temelju agilnog pristupa, razvijeno je mnogo metodologija, a od kojih su najpoznatije Ekstremno programiranje i Scrum.

10.1. Ekstremno programiranje

Ekstremno programiranje (skraćeno XP) jest metodologija temeljena na agilnom pristupu koja potiče izradu kvalitetnih proizvoda unatoč čestim promjenama zahtjeva klijenata [22]. Ekstremno programiranje orijentirano je na manje timove, obično do 20 članova, i na njihov timski rad. Članovi tima smješteni su u jednu sobi kako bi komunikacija bila najbolja, što omogućava bržu razmjenu informacija, lakše rješavanje problema i veću suradnju. Jedna od glavnih prednosti agilnog pristupa jest konstantna komunikacija s klijentima i ekstremno programiranje „poštuje“ to obilježje. Cilj ekstremnog programiranja jest isporučiti manje dijelove softvera u što kraćem vremenu pa se nove promjene u zahtjevima mogu lakše dodati [28].

Ekstremno programiranje često se bira kao metodologija izrade softvera jer ubrzava implementaciju, a vrijeme koje odlazi na čitanje i pisanje dokumentacije, dizajniranje dizajna i slične aktivnosti. Naravno, ovo samo vrijedi ako je tim manji jer je tada isplativije objasniti nego napisati, a zatim pročitati dokumentaciju [28].

Principi ekstremnog programiranja u velikoj se mjeri potpuno podudaraju s principima agilnog pristupa. Postoji 5 principa: komunikacija, jednostavnost, povratne informacije, poštovanje između svih dionika te hrabrost. Princip *hrabrosti* je tu potencijalno nejasan, a govori da kada se pojavi neki problem, ostali principi se moraju uzeti u obzir kako taj problem ne bi naštetio timu [28].

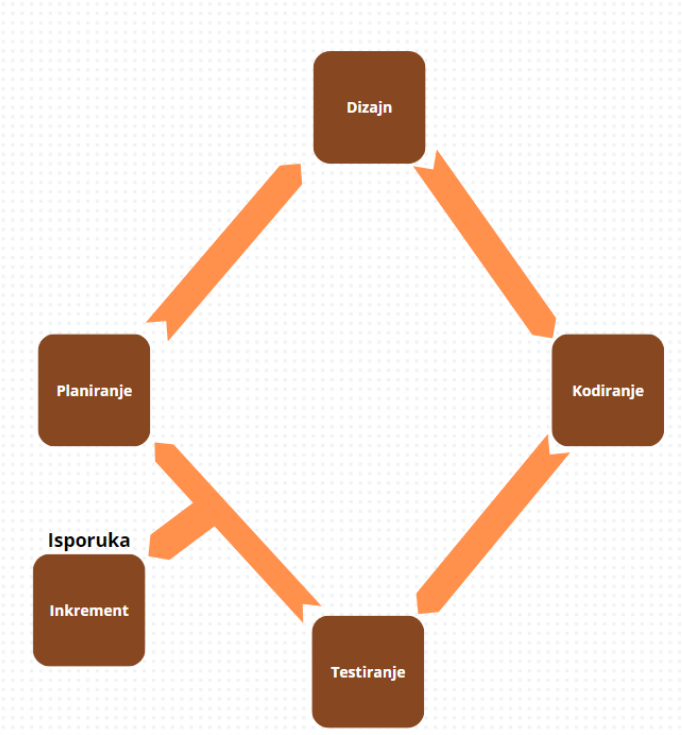
Ekstremno programiranje podrazumijeva veliku količinu praksi koje bi se trebale poštivati. One su podijeljene u 2 kategorije: vezane za zadovoljstvo klijenata i vezane za kvalitetu softvera.

Prva praksa vezana za zadovoljstvo korisnika jest osiguranje prisutnosti klijenata na licu mjesta, odnosno osigurati da predstavnici klijenata budu dio tima. Druga i posljednja praksa odnosi se na male isporuke proizvoda.

Postoji 6 praksi vezanih za kvalitetu softvera. Prva praksa jest korištenje metafora koje mogu pojasniti probleme. Zatim se 4 prakse fokusiraju na programiranje. Prva se odnosi na testiranje; kako programeri pišu kod, tako bi klijenti trebali testirati taj kod. Druga se odnosi na jednostavnost dizajna uz što manje klasa, metoda, varijabli itd. Treća praksa vezana za programiranje jest refaktoriranje koda kako bi ostao održiv kroz vrijeme. Posljednja praksa se odnosi na način programiranja, a to jest programiranje u paru. Kod takvog programiranja, jedan

programer piše kod, dok drugi razmišlja što bi ovaj prvi trebao pisati. Na taj se način kod može poboljšati, brže napisati, a programeri će biti zadovoljniji. Posljednja praksa iz kategorije kvalitete softvera jest kvalitetno upravljanje projektom koje obuhvaća planiranje, postavljanje prioriteta, procjene rokova isporuke proizvoda i postavljanje raznih standarada [28].

Na slici 9, prikazan je ciklus izrade softvera koristeći metodologiju ekstremnog programiranja. Postoje 4 koraka koja se izvršavaju iterativno, a to su planiranje, dizajn, kodiranje i testiranje. Testiranje je posljednja faza iteracije i nakon nje se isprogramirani dio softvera isporučuje, a zatim se ulazi u novu iteraciju koja započinje planiranjem [29].



Slika 9. Ekstremno programiranje (prema I. Purnami, 2023.)

Iz pruženog opisa ekstremnog programiranja, jasno se iščitavaju prednosti te metodologije. Zbog fleksibilnosti, efikasnosti, niske razine rizika od neuspjeha projekta i osiguranja dobre komunikacije i suradnje, ova metodologija je popularna i često se koristi. Ona stvara dobru atmosferu u organizaciji zbog načina na koji se programira (programiranje u paru), što rezultira velikom efektivnosti timova.

Ekstremno programiranje za sobom donosi i neke bitne probleme. Glavni nedostaci ekstremnog programiranja vide se u samoj srži ove metodologije, a to je da je naglasak na kodiranju, a ne na dizajnu i dokumentaciji. To je problem agilnih pristupa općenito i ekstremno

programiranje također ima isti problem, ali opet, nije niti to potpuno negativno jer ako se smanji dokumentacija, povećati će se brzina programiranja i na kraju, isporuka rješenja će biti brža. Također, tu su bitni i mali timovi koji konstantno komuniciraju pa dokumentacija i nije toliko potrebna kao kod nekih drugih, tradicionalnih modela. Tu se javlja drugi problem, a to je da tim ljudi mora biti na jednoj lokaciji u jednom prostoru. Posljednja stvar jest stres koji zahvaća zaposlenike. Često se programeri susreću s visokim pritiscima zbog budžeta i rokova što stvara veliki stres, no to je problem za kojega se može reći da se može pojaviti kod svih metoda [28].

10.2. Scrum

Scrum je metodologija koja se svrstava u agilni pristup izrade softvera, a cilj je osigurati visoku produktivnost u rješavanju kompleksnih zadataka kako bi se isporučili proizvodi visoke kvalitete [30]. Kao i kod ekstremnog programiranja, Scrum se temelji na sprintovima koji traju od 1 do 2 tjedna. Scrum ima jasno definirane uloge i odgovorenosti koje se nikada ne mijenjaju [22].

Scrum tim je fleksibilan, samoorganizirajući tim i nije im potreban menadžer da upravlja timom. Stoga, članovi tima mogu biti istovremeno kreativni i produktivni. Tim se sastoji od nekoliko uloga – vlasnik proizvoda (eng. Product Owner), Scrum Master i Scrum razvojni tim.

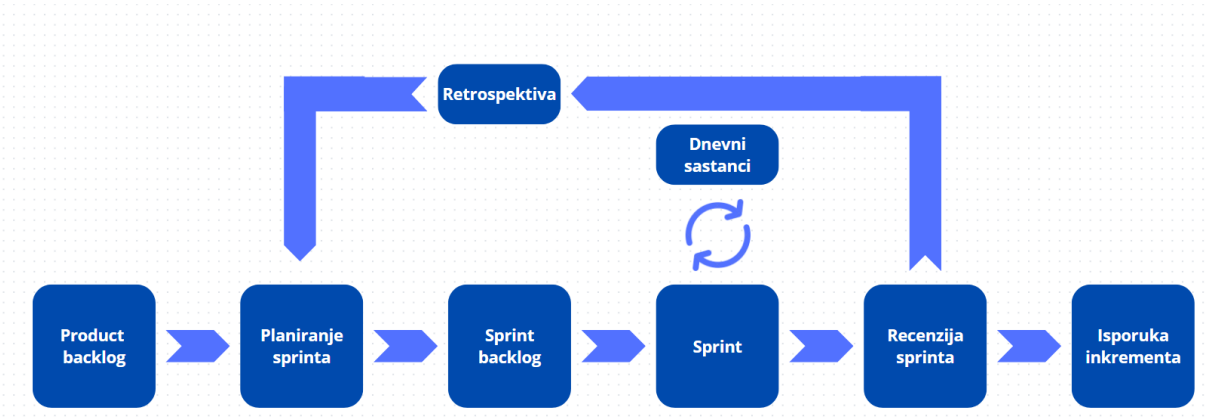
U timu, vlasnik proizvoda je samo jedan. Njegov zadatak jest osigurati kvalitetu proizvoda koji se isporučuje. Vlasnik proizvoda je zadužen za nekoliko bitnih zadataka od kojih se ističu sljedeći: dodavanje zadataka na backlog (lista nezavršenih zadataka), sortiranje tih zadataka, objašnjavanje zadataka timu i poboljšanje kvalitete rada. Što se tiče njegovog upravljanja backlogom, svi ostali djelatnici moraju poštovati njegove odluke.

Zadatak Scrum Mastera jest osigurati poštivanje metodologije i samo poznavanje metodologije Scruma od strane zaposlenika. Scrum se sastoji od pravila, vrijednosti i teorije i svi članovi tima bi trebali biti svjesni toga. Ako netko novi dođe u tim, Scrum Master ga upoznaje s radom. Preporučeno je da Scrum Master radi samo na jednom projektu kako bi se u potpunosti fokusirao na njega i, iako on nije menadžer, on bi trebao nastojati podizati učinkovitost tima.

Scrum razvojni tim je grupa profesionalaca (koja se prema [31] sastoji od 7 zaposlenika) koja je zadužena za isporuku novog proizvoda u obliku inkremenata. Neke zanimljivosti vezane za takav tim su da u timu ne postoje titule prema onome što rade, ne postoje podtimovi unutar jednog tima i iako su članovi specijalizirani samo za određene zadatke jezike i slično, oni moraju ostati fleksibilni kako bi mogli raditi i na ostalim područjima [30].

Model Scruma sastoji se od 2 backloga, planiranja sprinta, samog sprinta, recenzije i retrospektive sprinta te od dnevnih sastanaka. Jedan backlog odnosi se na proizvodne backlogove kojim upravlja vlasnik proizvoda i tu su sadržane korisničke priče funkcija koje se trebaju implementirati. Drugi backlog jest Sprint backlog i u njemu su sadržane sve stvari koje se planiraju obaviti tijekom jednog sprinta. Planiranje sprinta obuhvaća odabir stvari koje će se implementirati i u ovoj fazi komunicira se s vlasnikom proizvoda i dionicima o stvarima koje se trebaju isporučiti. Sprint jest faza u kojoj se izrađuju dogovorene stvari. Svaki sprint započinje sastankom i svakog dana održava se sastanak s ciljem da svi članovi tima vide kako sprint napreduje. Sprint traje kratko, između 2 i 4 tjedna s time da ne bi trebao trajati duže od mjesec dana. Nakon što je sprint gotov, potrebno je održati sastanak na kojemu se radi recenzija sprinta. Pošto Scrum kombinira inkrementalni i iterativni pristup, cijeli ovaj proces se ponavlja sve dok projekt nije gotov. Tijekom retrospektive, vodi se razgovor o problemima na koje se naišlo i o statusima zadataka; koji će biti dio sljedećeg sprinta, a koji neće [32].

Na slici 10, grafički je prikazan izgled Scrum metodologije [30].



Slika 10. Scrum (Prema scrum.org)

Korištenjem Scruma, organizacije mogu ostvariti niz prednosti koje nisu toliko prisutne u drugim modelima. Ovaj pristup omogućava timovima da budu učinkovitiji, fleksibilniji i više usmjereni na stvaranje vrijednosti za korisnike. Zadaci, sve informacije o njima i rezultati su jasno objašnjeni i vidljivi nakon svakog sprinta, a zbog konstantnih provedbi sastanaka, svi članovi tima znaju uglavnom sve o napretku projekta. Također, sprintovi povećavaju kvalitetu softvera koja se na kraju svakog sprinta može jasno procijeniti jer su rezultati vidljivi. Rizik od neuspjeha projekta je puno manji nego kod tradicionalnih modela razvoja softvera [32]. Tu se javljaju i ostale prednosti kao što su mogućnosti povratne informacije klijenata, timovi mogu biti kreativni i raditi samostalno, Scrum dobro raspoređuje budžet i vrijeme te garantira da će se svi zadaci obaviti kako treba [31].

Zbog brzine kojom programeri moraju isporučivati softver, može doći raznih problema. Na primjer, tu se javlja manja kvaliteta softvera i koda jer se radi užurbano. Problem može biti i moguće neznanje oko Scruma općenito. Prema rezultatima ankete H. Majeeda [33], čak 50% djelatnika nema znanja o Scrum procesu i nisu prošli formalni trening, već sve što znaju jest znanje stečeno od drugih kolega i Scrum Mastera. Problem u vezi Scrum Mastera je što može raditi samo na jednom projektu. Ako već radi na više projekata, to zahtjeva puno vremena i koordinacije. Problem može biti i stalna komunikacija u timu. S ciljem da se efikasnost poveća stalnim sastancima i razgovorima, može doći do kontraefekta i programeri će biti manje produktivni. Na posljetku, Scrum se previše oslanja na idealne uvjete. Prema Scrumu, programeri bi trebali znati sve kako bi bili fleksibilni, trebali bi biti brzi i trebali bi se samostalno organizirati, a u realnosti, to često nije slučaj [33]. Sličan problem koji se javlja i kod ekstremnog programiranja jest izostanak dokumentacije.

Razlike između Scruma i tradicionalnih modela, na primjer vodopadnog ili inkrementalnog su jasno vidljive, no razlika između Scruma i druge metodologije agilnog pristupa, ekstremnog programiranja, puno su teže uočljive. Razlike između ove dvije metodologije su vidljive u vremenskim rokovima, načinu rada i fleksibilnosti. Kod ekstremnog programiranja, vremenski rokovi su fleksibilniji i mogu se mijenjati, dok Scrum ne dopušta nikakve promjene u vidu rokova ili bilo kakvih drugih smjernica. Ekstremno programiranje temelji se metodologijama softverskog inženjerstva, dok je Scrum nešto fleksibilniji po tome pitanju jer dopušta samoorganizaciju u timu. Također, u Scrumu tim, tj. Scrum Master odlučuje prioritete zadataka, a kod ekstremnog programiranja, tim to ne može raditi jer su prioriteti zadataka već predefinirani.

11. Usporedba tradicionalnih modela s metodologijama agilnog pristupa

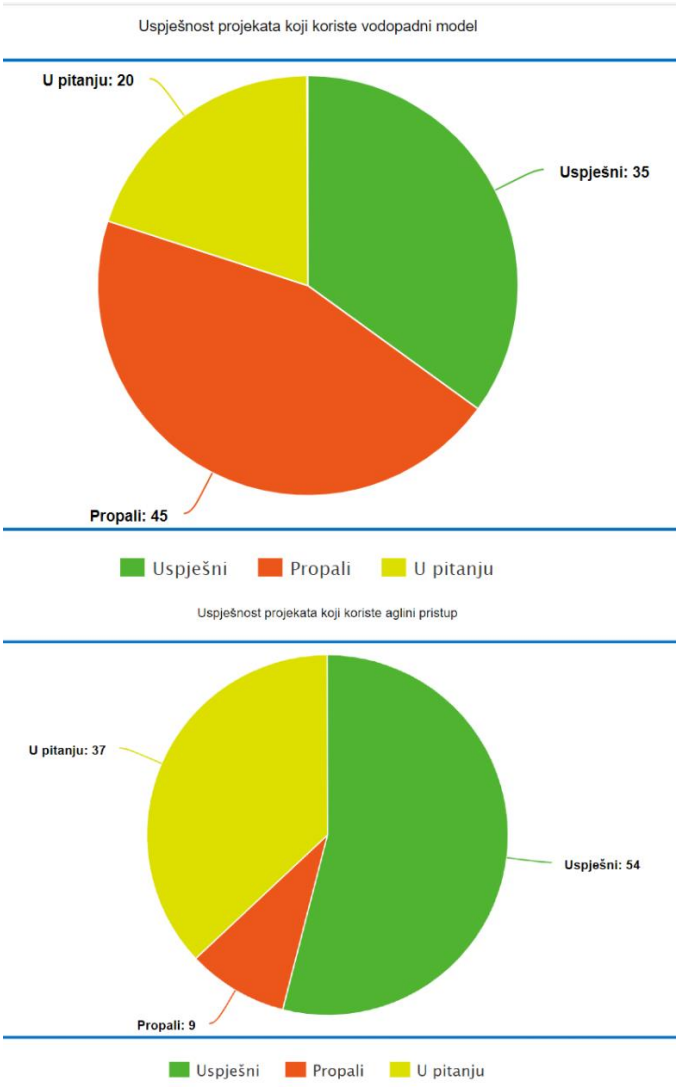
Pod tradicionalne modele svrstavaju se vodopadni, inkrementalni i spiralni model. Agilni pristup obuhvaća ekstremno programiranje i Scrum. To nisu svi tradicionalni modeli, odnosno metodologije agilnih pristupa. Postoje još mnoge druge metode, no ovo su neki od najpoznatijih i najkorištenijih modela/metodologija. U tablici 1 prikazane su razlike između tradicionalnog i agilnog razvoja softvera (tablica je uglavnom stvorena prema [25]).

Tablica 1. Usporedba tradicionalnih modela i agilnih pristupa razvoju softvera (prema Stoici et al., 2013.)

	Tradicionalni modeli	**Agilni pristup**
Veličina projekta	Preferiraju se manji projekti	Najbolje koristiti za velike i dugotrajne projekte
Kompleksnost projekta	Jednostavniji projekt	Složeni projekt
Promjenjivost zahtjeva	Ovisno o metodi, zahtjevi moraju biti stabilni i ne smiju se mijenjati	Nema problema kod izmjene zahtjeva
Kasnije dodani zahtjevi	Ovisno o metodi, može doći do problema ako se zahtjevi dodaju kasnije	Nema problema kod dodavanja novih zahtjeva
Organizacijska struktura	Birokratska, strogo upravljana struktura, tipično kod većih organizacija	Organska, samoorganizirajuća struktura, tipično kod manjih organizacija
Veličina tima	Veliki timovi	Manji timovi
Komunikacija u timu	Komunikacija nije je u prvom planu	Svi dionici uključeni u izradu konstantno komuniciraju
Lokacija programiranja	Lokacija nije toliko bitna, moguć je i rad od kuće	Izrazito bitna, poželjno da svi članovi tima budu u jednoj prostoriji
Komunikacija s klijentima	Ovisno o modelu, nije u prvom planu	Veliki fokus na komunikaciju s klijentima
Uključenost klijenata u proces razvoja	Niska uključenost klijenata	Visoka uključenost klijenata
Kontrola kvalitete	Stroga kontrola s poteškoćama u planiranju	Konstantna kontrola kvalitete

Trošak kretanja ispočetka	Veliki trošak	Nizak trošak
Testiranje softvera	Testiranje je posebna faza nakon programiranja	Testiranje nakon svake iteracije
Glavni cilj metoda	Postići visoku razinu sigurnosti	Pružiti brzu vrijednost klijentima
Rizik od neuspjeha	Visok rizik od neuspjeha	Nizak rizik od neuspjeha
Dizajn	Veći fokus na dizajn	Manji fokus na dizajn
Dokumentacija	Veći fokus na dokumentaciju	Manji fokus na dokumentaciju

Razlika između tradicionalnih, odnosno konkretno vodopadne metode i metoda agilnog pristupa vidljiva je na grafičkom prikazu koje je nastalo na temelju istraživanja prema [31].



Slika 11. Usporedba uspješnosti vodopadnog modela s agilnim pristupom

Šanse da projekt uspješno završi puno su veće za agilne metodologije. Ako se koriste one, na primjer Scrum ili ekstremno programiranje, šanse da projekt bude uspješan su 54%, dok samo 9% projekata propadne. Za vodopadni model, šanse da projekt završi uspješno bez ikakvih problema su samo 35%, a čak svaki peti projekt završava bez uspjeha.

12. Specifične metodologije za izradu aplikacija prividne stvarnosti

Pomoću vodopadnog, inkrementalnog, spiralnog i drugih tradicionalnih modela te pomoću metodologija agilnog pristupa, moguće je izraditi sve vrste softvera – desktop, mobilne, web aplikacije pa tako i aplikacije prividne stvarnosti. Kako popularnost prividne stvarnosti raste, javlja se sve više i više specifičnih metodologija koje se fokusiraju samo na razvoj aplikacija prividne stvarnosti i konkretno opisuju njihov postupak.

12.1. TRES-D metodologija

TRES-D metodologija (punog naziva *ThREe dimensional uSer interface Development methodology*) nastala je jer su tradicionalni i agilni modeli preopćeniti, odnosno razvoj aplikacija prividne stvarnosti poprilično se razlikuje od razvoja standardnih aplikacija. TRES-D metodologija kombinira brojne druge metodologije namijenjene za razvoj interakcije između čovjeka i računa i njih spaja u jedan novi model koji kombinira iterativni i inkrementalni pristup [34].

TRES-D sastoji se od 6 aktivnosti. Prva aktivnost jest prikupljanje podataka o projektu od klijenta, identificiranje funkcionalnih i nefunkcionalnih zahtjeva, određivanje budžeta i rokova.

Druga aktivnost jest razumijevanje zahtjeva kako bi se dizajn mogao izraditi kvalitetno. Pomoću ove faze, analiziraju se zadaci, određuju se karakteristike aplikacije, dodjeljuju se zadaci članovima tima itd.

Treća aktivnost jest konceptualni dizajn aplikacije. U njoj dizajneri bilježe ideje i stvaraju prototipove za njih. Ovo je faza u kojoj sudjeluju i klijenti koji daju svoja mišljenja. Programeri i modeleri zaduženi su za utvrđivanje mogu li implementirati ideje i imaju li potreban softver ili hardver za to. Rezultat ove faze je izvješće s rješenjima, vremenskim rokovima, prednostima i nedostacima odabranog rješenja, hardveru i softveru koji će omogućiti implementaciju tih rješenja.

Iterativni dizajn jest 4. aktivnost u TRES-D metodologiji, a razlog zašto se zove „iterativni“ jest taj što dizajner kroz cikluse dizajnira svaku komponentu aplikacije zasebno. Iterativni se dizajn dijeli na dvije podaktivnosti: apstraktni dizajn i prezentacijski dizajn. Apstraktni dizajn je dizajn visoke razine; on je neovisan o platformi i fokusira se na dizajn svijeta, objekata, komunikaciju između objekata i igrača. Prezentacijski dizajn, za razliku od apstraktnog dizajna, uključuje i hardver i softver. Dizajnira se izgled i struktura svijeta te objekata, geometrija i

slično. Ovaj dizajn uključuje i razradu interakcija za svaku grupu operacija koje su identificirane u apstraktnom dizajnu, a operacije su funkcije koje određeni objekti sadrže.

Nakon dizajna, slijedi implementacija rješenja. Tu se kreiraju 3D objekti, izrađuju se animacije i skripte i sve ostale stvari koje su potrebne za izradu virtualnog svijeta.

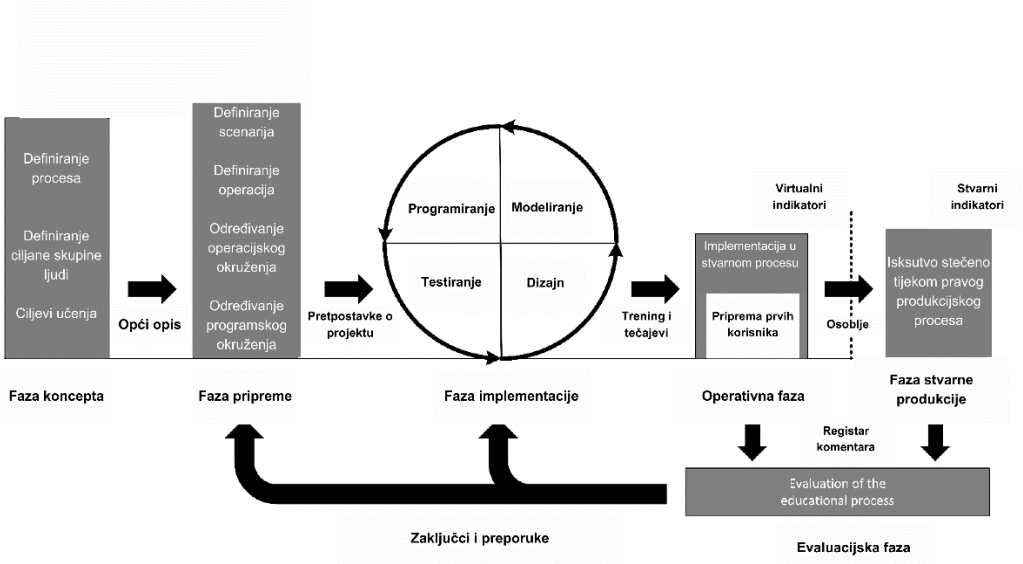
Posljednja aktivnost ove metodologije jest isporuka aplikacije i njezino održavanje [34].

TRES-D metodologija korisna je jer točno definira alate koji će se koristiti, precizno raspoređuje zadatke prema ulogama i veliki fokus stavlja na dizajn prije samog programiranja. Korisna je i za eksperte, ali i za one kojima je razvoj aplikacija prividne stvarnosti samo hobi [34].

12.2. Metodologija za izradu aplikacija obrazovanja u Industriji 4.0

Industrija 4.0, odnosno 4. industrijska revolucija, automatizirala je i digitalizirala sve procese vezane za proizvodnju i logistiku - promijenila je cijelo tržište i način rada. Zbog toga, nužno je poboljšati edukaciju. Jedno od novijih i ne toliko istraženih područja je prividna stvarnost, koja ima veliki potencijal za poboljšanje edukacijskog procesa jer kroz imerziju i stimuliranje osjetila može poboljšati učenje [35].

Metodologija koju su razvili A. Paszkiewicz et al. [35] dizajnirana je za izradu raznih trening tečajeva. Sastoji se od 6 faza: konceptualne faze, faze pripreme, implementacije, operativna i evaluacijske faze te stvarne produkcijske faze.



Slika 11. Metodologija za obrazovanje (Prema Paszkiewiczu et al., 2021.)

Tijekom faze koncepta, određuju se osnovne i početne stavke projekta, a to su ciljevi, zahtjevi, očekivanja i karakteristika budućih korisnika aplikacije. Utvrđuju se i ciljevi učenja koji ne smiju biti kompleksni, a moraju osigurati stjecanje znanja. Rezultat ove faze jest dokument u kojem je opisan opći plan treninga [35].

U fazi pripreme, potrebno je stvoriti scenarij korištenja aplikacije. To uključuje operacije koje će korisnici raditi, kako će sustav reagirati na te operacije, odnosno kako će reagirati ako se dogodi greška. Potrebno je izraditi i popis objekata zajedno s njihovim opisima koji će biti dio virtualne okoline i s kojima će se moći komunicirati. Rezultat ove faze su pretpostavke o projektu [35].

Faza implementacije dijeli se na 4 aktivnosti: dizajn, modeliranje, programiranje i testiranje. Tijekom dizajna, odlučuje se za koje će se objekte provoditi modeliranje, a za koje samo izrada i postavljanje teksture. Definiraju se i neke dodatne stvari kao što su pitanja sigurnosti i pouzdanosti, različite kontrole scenarija, funkcije okruženja i slično. Rezultat dizajna su tehnički dizajn i implementacijska dokumentacija. U aktivnosti modeliranja, provodi se modeliranje trodimenzionalnih objekata i pripremaju se teksture za objekte koji će se nalaziti u okruženju. Modeliranje se može izvršavati kao samostalna aktivnost, no može se raditi paralelno sa programiranjem. Programiranje rezultira funkcionalnim virtualnim okruženjem. Posljednja aktivnost tijekom implementacije je testiranje rješenja. Cilj testiranja jest utvrditi je li rješenje prikladno i zadovoljava li potrebe korisnika koji također moraju biti uključeni u testiranje [35].

Operativna faza služi kako bi korisnici naučili koristiti aplikaciju i kako bi se pripremili za stvarne situacije. Ovdje je cilj osigurati da će se aplikacija koristiti na način na koji je to zamišljeno. Potrebno je provesti testove i u ne produkcijskom okruženju i u produkcijskom kako bi se dobili potrebni podaci o efektivnosti aplikacije, odnosno treninga. Cilj ove faze je prikupiti sve komentare i prijedloge koje korisnici imaju kako bi se mogla izvesti faza evaluacije. Poželjno je da komentiraju svi dionici koji su povezani s aplikacijom – obični korisnici, treneri (administratori), dioničari itd. Faza evaluacije koristi te komentare kako bi se razvojni proces vratio na fazu pripreme ili implementacije, ovisno o tome kakvi su komentari [35].

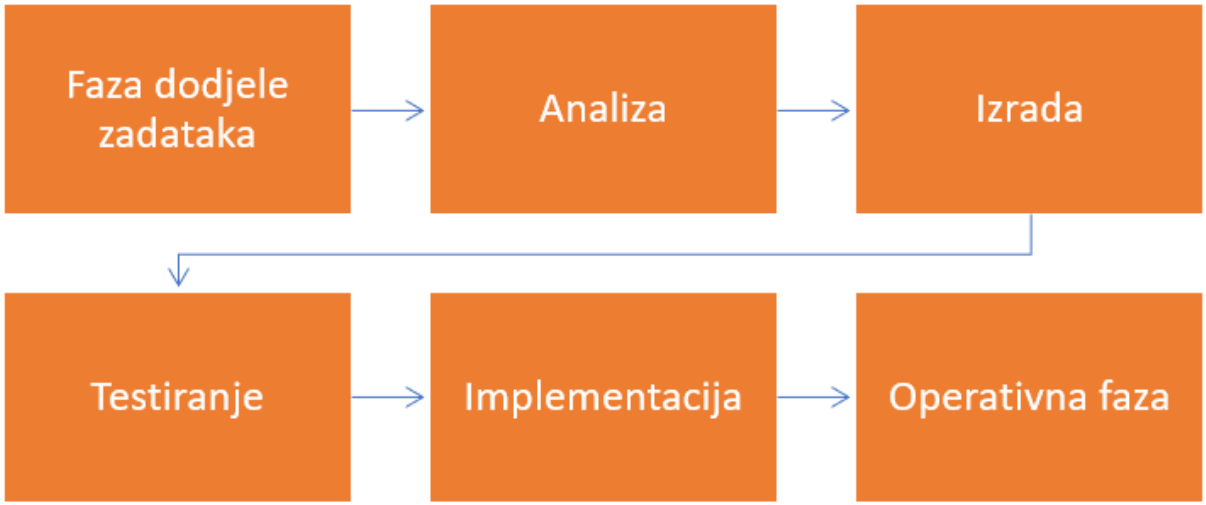
Korištenje ove metodologije donosi mnoge benefite tijekom razvoja aplikacija prividne stvarnosti. Nju je dobro primjenjivati za projekte svih veličina, bilo da su mali, srednji ili veliki, no trajanje tih projekata bi trebalo biti kratko. Ova metodologija je najprikladnija za primjenu u raznim područjima industrije i obrazovanja, uključujući različite vrste treninga, edukacije i programa. Metodologija koristi spiralan pristup izradi aplikacija, što je dobro jer omogućuje lakše izmjene postojećih i dodavanje novih funkcionalnosti, a to znači da je metodologija iznimno fleksibilna. Važna stavka jest i komunikacija s osobama uključenima u razvoj softvera,

jer ta suradnja može dovesti do znatno kvalitetnijeg i funkcionalnijeg konačnog proizvoda. Važno je spomenuti i dokumentaciju; svaka faza u razvoju aplikacije prividne stvarnosti je dokumentirana [35].

Za razliku od TRES-D metodologije, ova metodologija je puno specifičnija i odnosi se na izradu rješenja u sklopu industrije i treninga. Druga razlika vidljiva je i u pristupu izradi projektu. TRES-D koristi kaskadni pristup gdje se svaka faza odvija samo jednom. Metodologija obrazovanja za Industriju 4.0 fleksibilnija je metoda jer koristi spiralni pristup te se postupak izrade aplikacije može ponavljati nakon kraja faze evaluacije.

12.3. Metodologija za izradu aplikacije prividne stvarnosti u Unityju 3D

Metodologija za izradu aplikacija prividne stvarnosti u Unityju 3D nastala je zbog manjka kvalitetnih, a specifičnih metodologija za izradu aplikacija prividne stvarnosti. Također, ona se fokusira na popularno i besplatno okruženje Unity koje je lagano za korištenje, a omogućuje kompatibilnost s mnogo različitih uređaja. Metodologija se sastoji od 6 faza: faza dodjele zadataka, faza analize, izrade, testiranja, implementacije i operativna faze. One se odvijaju kaskadno, jedna nakon druge bez mogućnosti povratka na prijašnje faze. Metodologija je vidljiva na slici 12 [36].



Slika 12. Unity 3D metodologija (Prema J. Polcaru et al., 2016.)

U prvoj fazi definiraju se zahtjevi klijenata. Kroz zahtjeve, potrebno je izvući cilj aplikacije što je iznimno važno kako bi se razumjelo zašto se uopće aplikacija izrađuje. Zatim je potrebno definirati scenarij koji će odrediti priču, zadatke, ugođaj i interakciju u virtualnom

svijetu. Potrebno je odrediti i hardver koji će se koristiti te odrediti razinu imerzije (nije uvijek potrebno ostvariti najveću moguću imerziju). Definiranje ciljne skupine korisnika također se odvija u ovoj fazi [36]. Bitno je znati tko će koristiti aplikaciju; jesu li to djeca, starije osobe, studenti, polaznici sigurnosnog tečaja i slično jer ovisno o skupini ljudi, morati će postojati razlike u dizajnu korisničkog sučelja, virtualnog okruženja i načina na koji su funkcije kreirane. Na primjer, ako će aplikaciju koristiti djeca, veće su šanse da će korisničko sučelje biti šareno i imati će mogućnost kretanja. Ako su korisnici starije osobe, bilo bi dobro da su elementi korisničkog sučelja veći i da se korisnik može kretati bez da mora stajati u stvarnome svijetu [36].

U fazi analize potrebno je analizirati zaključke koji su doneseni u prvoj fazi. Ovdje je potrebno biti posebno oprezan jer će svaka greška biti kasnije izrazito skupa i naštetiti će daljnjem razvoju aplikacije. U suradnji s klijentom, potrebno je detaljizirati scenarij tako što se napravi popis svih zadataka, akcija i priča. Izrađuje se i lista objekata i stanja (npr. kretanje/stajanje na mjestu), a zatim se stvaraju klase na temelju sličnih objekata. Akcije su okidači koje je potrebno dodijeliti svakom objektu. Kada se dogodi akcija, objekt mijenja stanje. Ovo je potrebno prikazati pomoću UML dijagrama promjene stanja [36].

Faza izrade slična je ekvivalentnoj fazi kod ekstremnog programiranja. Ovdje se cijelo vrijeme komunicira s klijentima kako bi se kvalitetno izradili *asseti*, odnosno resurse koji će se koristiti za izgradnju virtualnog okruženja. *Asseti* su 3D objekti, skripte, grafike, animacije, zvučni efekti i svi ostali elementi koji se koriste za izgradnju aplikacije u Unityju. Skripte su ovdje od velikog značaja jer one „oživljavaju“ svijet. Pomoću njih se pokreću animacije, mijenjaju se stanja objekata, postavljaju se okidači i slično.

Faza testiranja dolazi nakon faze izrade, no svojevrsno testiranje se odvija konstantno. Potrebno je prvenstveno testirati rade li skripte ispravno, a zatim se provjeravaju ostali resursi. Iako se često testiranje povezuje s pronalaskom i rješavanjem grešaka, ovdje je potrebno testirati i doživljaj u virtualnom svijetu. Testiranje se također izvodi zajedno s klijentima kako bi oni mogli utvrditi ide li razvoj u pravome smjeru [36].

U fazi implementacije aplikacija se isporučuje zajedno s hardverom. Potrebno je obaviti kalibraciju za svjetlo i akustiku kako bi interakcija bila kvalitetna. Posljednje testiranje se odvija u ovoj fazi [36].

Operativna faza je posljednja faza i u njoj u kojoj se prikupljaju podaci tijekom korištenja aplikacije. Podaci se mogu prikupljati preko dnevnika (eng. *log*) u koji se zapisuju bilo kakvi problemi s kojima se korisnici susretnu. Na temelju prikupljenih podataka, kasnije se rade dorade aplikacije i ispravljaju se greške. U ovoj fazi se također komunicira s klijentima;

razgovara se o stečenom iskustvu i potencijalnoj mogućnosti ponovnog korištenja resursa za neke druge projekte [36].

Koliko je ova metoda zapravo jaka, dokazano je na nekoliko projekata. *Edutainment* igra i virtualna radionica na Fakultetu mehaničkog inženjerstva, verzija Minnesota Dexterity Testa i vizualizacijski alat za vizualizaciju podataka iz LiDAR-a neki su od većih projekata na kojima se ova metodologija pokazala uspješnom [36].

Na temelju slike 12, moglo bi se pomisliti da je ovo jednostavna metodologija, no taj dijagram je samo jako pojednostavljen. Iako se faze odvijaju kaskadno, ova metodologija je pogodna i za veće projekte jer se testira konstantno i komunikacija s klijentima je stalna. Zbog toga, prostor za greške je iznimno mali, ali zato svaka greška je izrazito skupa ako se ne otkrije na vrijeme. Kako se navodi u [36], ova metodologija se koristi kod okruženja koja sadrže puno interakcije između svijeta i korisnika pa je odlična za razvoj igara, a igre su najteže za izraditi jer moraju imati visoku razinu imerzije koja se postiže kroz ugodno virtualno okruženje u kojem se može komunicirati s mnogim stvarima. Što je najvažnije, ova metodologija je svoju snagu dokazala u praksi i pokazala se kao jaka alternativa za starije i „formalnije“ metodologije.

12.4. Metodologija za stvaranje afektivnog VR iskustva

Dobro dizajnirana virtualna okruženja mogu izazvati određene emocije kod korisnika. To je teško postići, a otežava i činjenica da jako dugo vremena nije postojala nikakva metodologija koja bi se fokusirala na afektivni dio virtualnog iskustva. Stoga, okupili su se mnogi stručnjaci [37] kako bi razvili posebnu metodologiju koja se fokusira na izazivanje 5 emocionalnih reakcija – sreća, tuga, strah, gađenje i ljutnja.

Metodologija za stvaranje afektivnog VR iskustva sastoji se od 3 faze: analiza emocija u stvarnome svijetu, stvaranje VR okruženja i procjena emocija [37].

Analizom emocija u stvarnome svijetu želi se odabrati i istražiti emocije koje će biti dio virtualnog iskustva. Ovdje je potrebno odgovoriti na pitanja kao što su zašto baš tu emociju biramo, tko su korisnici i u kakvom se oni okruženju nalaze i kojim mehanizmima će virtualno okruženje izazvati tu emociju [37]. U tablici 2, prikazani su mehanizmi i emocije koje oni izazivaju.

Tablica 2. Mehanizmi povezani s emocijama

Mehanizam	**Izazvana emocija**
Opasnost	Strah
Frustracija	Ljutnja
Ugodni doživljaj	Sreća
Samoća	Tuga
Trulost (tekstura)	Gađenje

Nakon što se odaberu emocije, slijedi faza stvaranja VR okruženja. Prvo je potrebno definirati elemente dizajna, a zatim se izrađuje svijet. Elemente je potrebno otkriti kroz postavljanje nekoliko pitanja: koji su to elementi koji će izazvati emocije, kakva bi mogla biti priča kod doživljaja korisnika i kako povezati te sve elemente u jednu cjelinu. Elementi dizajna su semantički, senzorni, dinamički i interaktivni. Semantički elementi su lokacije koje će uzrokovati određene emocije, senzorni elementi su podražaji koji će prenijeti semantička značenja, dinamički elementi su sve promijene okruženja, a interaktivni elementi su objekti s kojima korisnici interaktiraju. Oni se spajaju u jednu cjelinu kako bi se postiglo bogato okruženje [37].

Procjena emocija zadnja je faza izrade afektivnog VR iskustva. Ona se sastoji od 3 aktivnosti: testiranja, podešavanja i validacije. Testiranjem se otkrivaju stvari koje nisu dobre, a zatim se one moraju podesiti. Testiranje se provodi na temelju prototipa kojega testira ograničen broj ljudi prije same isporuke. Ovaj proces testiranja i podešavanja je iterativan i izvršava se sve dok novi problemi neće biti uočeni. Na kraju slijedi validacija u kojoj sudjeluje više ljudi i tu se koriste različite tehnike procjene kao što su upitnici i tehnologije za emocionalnu procjenu (npr. prepoznavanje izraza lica) [37].

Od svih metodologija fokusiranih na prividnu stvarnost, a obrađenih u ovome radu, ova metodologija je najspecifičnija jer opisuje proces izrade posebne aplikacije prividne stvarnosti koja se temelji na osjećajima. Kvaliteta ove metodologije vidljiva je jer se je koristila za izradu nekoliko različitih projekata.

Kod ove metodologije javljaju se i neka ograničenja. Virtualna okruženja testirana su samo u polu-imerzivnoj tehnologiji, no predviđa se da bi osjećaji kod veće imerzije bili još jače prisutni. Drugi problem jest što emocije nastaju zbog okruženja, a ne zbog metodologije, stoga se ne može reći da je ova metodologija dobra, odnosno da radi [37].

13. Izrada jednostavne aplikacije prividne stvarnosti pomoću modela vodopada

Na temelju izrade jednostavne aplikacije prividne stvarnosti pomoću modela vodopada, bit će prikazan i dokumentiran cijeli proces kako bi se praktično uvidjele prednosti i nedostaci ovog modela. Aplikacija će biti virtualna galerija i u njoj će korisnici moći pregledavati slike prema različitim razdobljima umjetnosti – konkretno, bit će 3 razdoblja. Prvo razdoblje je renesansa i ona će biti odmah otključana. Čim korisnik riješi kviz o renesansnim slikama koje se nalaze u galeriji, otključati će novo razdoblje. Drugo razdoblje bit će impresionizam, a treće barok.

Aplikacija će biti izrađena u motoru (eng. Game Engine) Unity koji je besplatan, a omogućuje izrazito jednostavnu i brzu implementaciju aplikacija koje se temelje na prividnoj stvarnosti.

13.1. Analiza zahtjeva

Osim što će korisnici moći vidjeti slike, moći će slušati zanimljivosti o samoj slici. Na kraju svake sobe (jedna soba predstavlja jedno razdoblje umjetnosti), korisnici će moći riješiti jednostavan kviz kako bi otključali nova razdoblje. Na taj način bi se osiguralo da su korisnici stvarno poslušali informacije. Kada korisnici uspješno riješe kviz, bit će im omogućen ulazak u sobu, a u sobe će ulaziti tako što će pritisnuti gumb za otvaranje vrata.

Unity nudi 2 vrste kretanja: teleportacija i kontinuirano kretanje. Odabrano je kontinuirano kretanje jer je to imerzivniji način kretanja. Korisnik se također može okretati na 2 načina: kontinuirano i za određeni kut (eng. *snap*). Pošto je već kretanje kontinuirano, odabrano je i kontinuirano okretanje.

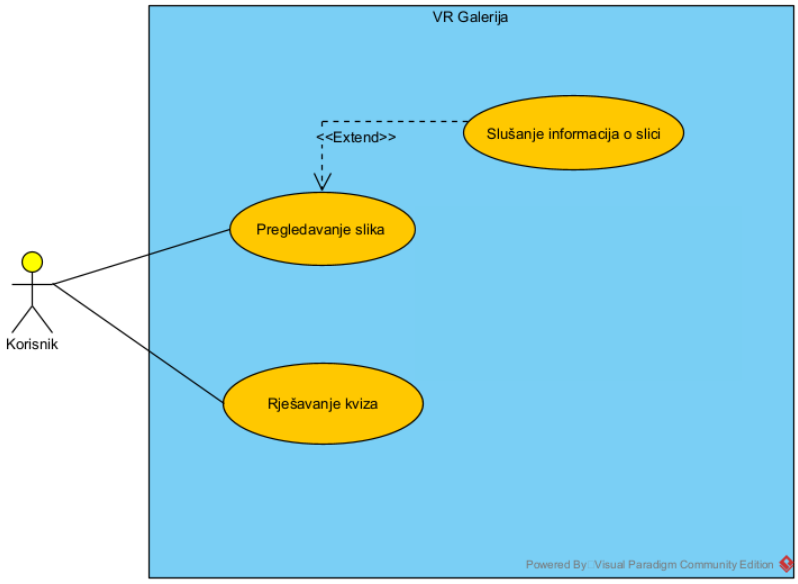
Kada korisnici izađu iz aplikacije, sva vrata koja su otključali moraju biti otključana i kada opet otvore aplikaciju. Prema tome, aplikacija će morati spremati sav napredak kojega je igrač postigao tijekom sesije igranja. Spremanje napretka bi se trebalo obavljati lokalno kako bi se izbjeglo korištenje servera i baze podataka, odnosno ne bi postojala potreba za internetskom vezom.

U tablici 3 nalaze se specificirani funkcionalni zahtjevi.

Tablica 3. Funkcionalni zahtjevi

Identifikator	**Zahtjev**
FZ-1	Pregledavanje slika prema razdobljima umjetnosti (renesansa, impresionizam, barok)
FZ-2	Pružanje zvučnih informacija o slikama
FZ-3	Rješavanje kviza kako bi se otključala nova razdoblja umjetnosti
FZ-4	Otvaranje vrata mora biti animirano
FZ-5	Spremanje napretka

Funkcionalni zahtjevi mogu se prikazati i preko dijagrama slučajeva korištenja, a dijagram je prikazan na slici 13. Glavne aktivnosti su pregledavanje slika i rješavanje kviza nakon svakog razdoblja. Pregledavanje slika prošireno je kroz aktivnosti slušanja informacija o slici. Ne postoji obveza da se kviz može riješiti samo ako se sve posluša, stoga ove aktivnosti nisu povezane (kviz se može riješiti i bez pregledavanja slika ako korisnik zna odgovore). Kako bi uspješno riješio kviz, korisnik mora uzastopno točno odgovoriti na 3 postavljena pitanja.



Slika 13. Dijagram slučajeva korištenja

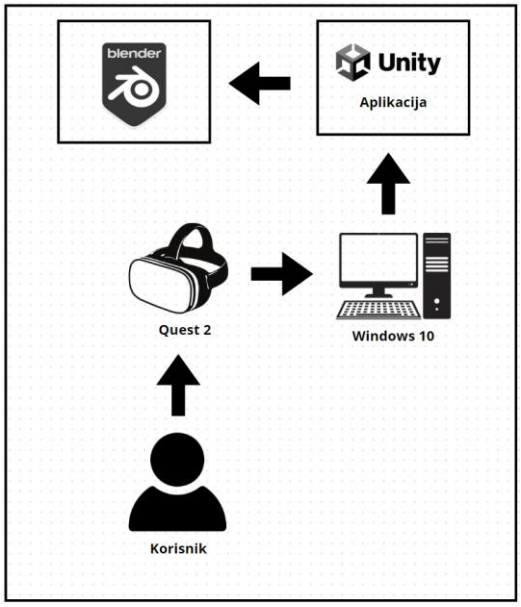
Nefunkcionalni zahtjevi, odnosno zahtjevi kvalitete, definiraju koliko dobro bi aplikacija trebala raditi. Oni se najčešće ne definiraju od strane klijenata, već se podrazumijevaju i njih mora definirati projektni tim. Svi identificirani nefunkcionalni zahtjevi vidljivi su u tablici 4.

Tablica 4. Nefunkcionalni zahtjevi

Identifikator	**Zahtjev**
NFZ-1	Performanse - aplikacija će se izvoditi glatko bez ikakvih trzanja
NFZ-2	Dostupnost - aplikacija bi trebala biti stalno dostupna jer se ne mora spajati na internet
NFZ-3	Kompatibilnost - mora raditi na uređaju Oculus Quest 2 tako što se pokrene na operacijskom sustavu Windows 10
NFZ-4	Kapacitet – igra ne bi smjela biti veća od 5 gigabajta

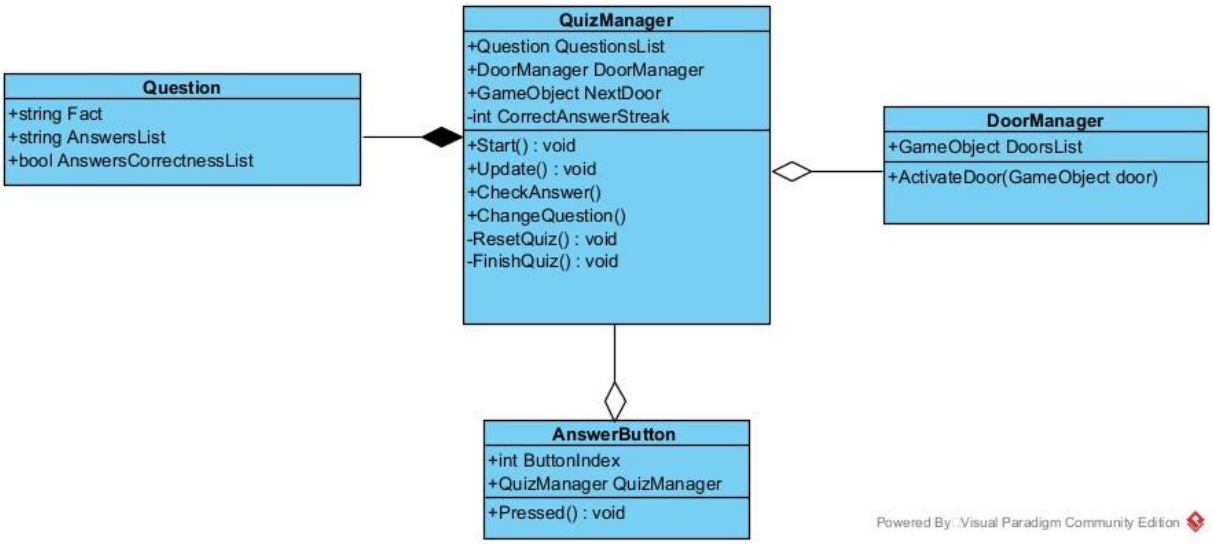
13.2. Dizajn sustava

Arhitektura aplikacije sastoji se od nekoliko uređaja i softvera. Prvo, korisnik koristi uređaj Oculus Quest 2 kako bi uopće mogao koristiti aplikaciju. Za pokretanje aplikacije koristiti se računalo s operacijskim sustavom Windows 10. Računalo je svojevrsno povezano s Unityjem u kojem je izrađena aplikacija. Modeliranje 3D objekata provodi se u softveru Blender. Kreirani objekti u Blenderu koriste se u Unityju za izgradnju virtualnog okruženja. Na slici 14, prikazan je dijagram koji prikazuje arhitekturu na najmanjoj razini detalja. Detaljniju verziju je teže prikazati jer Unity ima gotovi predložak za izgradnju VR aplikacija, a on koristi svoje pakete kako bi omogućio i olakšao izradu aplikacije prividne stvarnosti.



Slika 14. Arhitektura sustava

Kako bi se olakšala faza dizajna sustava, poželjno je kreirati dijagram klasa. U slučaju virtualne galerije, potrebno je stvoriti dijagram klasa koje su povezane s kvizom jer je to središnji dio aplikacije. Na slici 15, prikazan je taj dijagram.

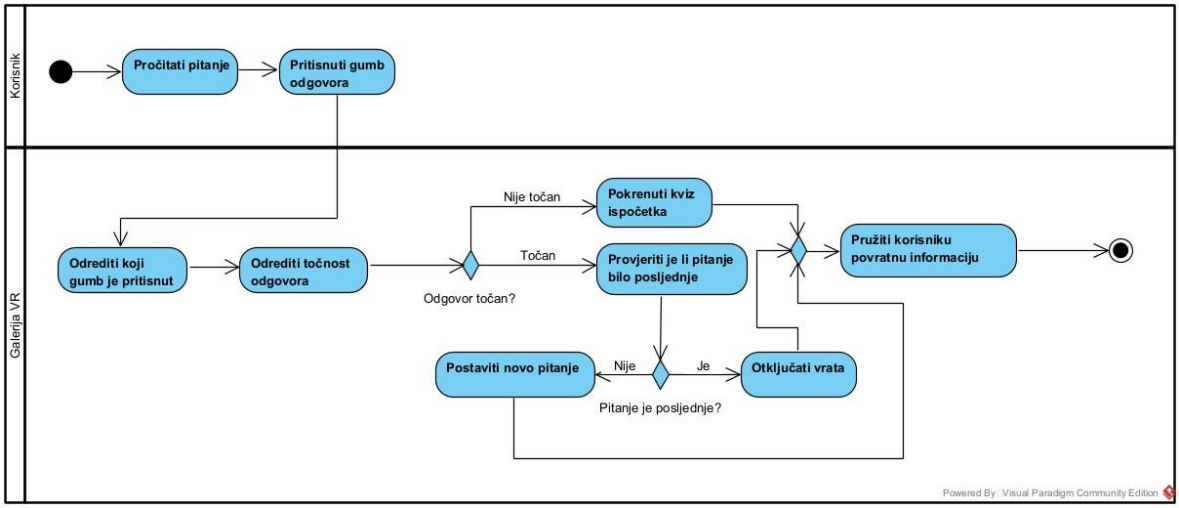


Slika 15. Dijagram klasa

Srž kviza je klasa *QuizManager.* Ona će upravljati kvizom – određivati će je li korisnik odgovorio točno, resetirati će kviz ako nije, pozvati će metodu za otključavanje vrata ako je odgovor točan i izmjenjivati će pitanja nakon svakog pruženog odgovora, odnosno pritiska na gumb koji se registrira u klasi *AnswerButton*.

Ovo nisu sve klase, već postoje i neke potpuno samostalne kao što je klasa za vrata – ona će zasigurno imati metodu koja će otvarati ta vrata. To su vrlo jednostavne klase pa ih nema potrebe stavljati u dijagram klasa. Potrebno je napomenuti da implementacija možda neće pratiti 100% ovaj dijagram klasa jer je teško predvidjeti sve u vezi implementacije.

Pomoću dijagrama aktivnosti na slici 16, prikazan je tok i komunikacija između korisnika i aplikacije. Aplikacija će morati utvrđivati nekoliko stvari kako bi kviz radio ispravno. Prvo mora odrediti koji gumb je pritisnut, a zatim je li odgovor koji je povezan s tim gumbom točan. Ako je točan, provjerava se je li to bilo posljednje pitanje i ako je, završava se kviz i otključavaju se vrata za novo razdoblje. Ako ima još pitanja, postavlja se novo pitanje.



Slika 16. Dijagram aktivnosti

U tablici 5, vidljive su ostale tehničke informacije vezane za projekt.

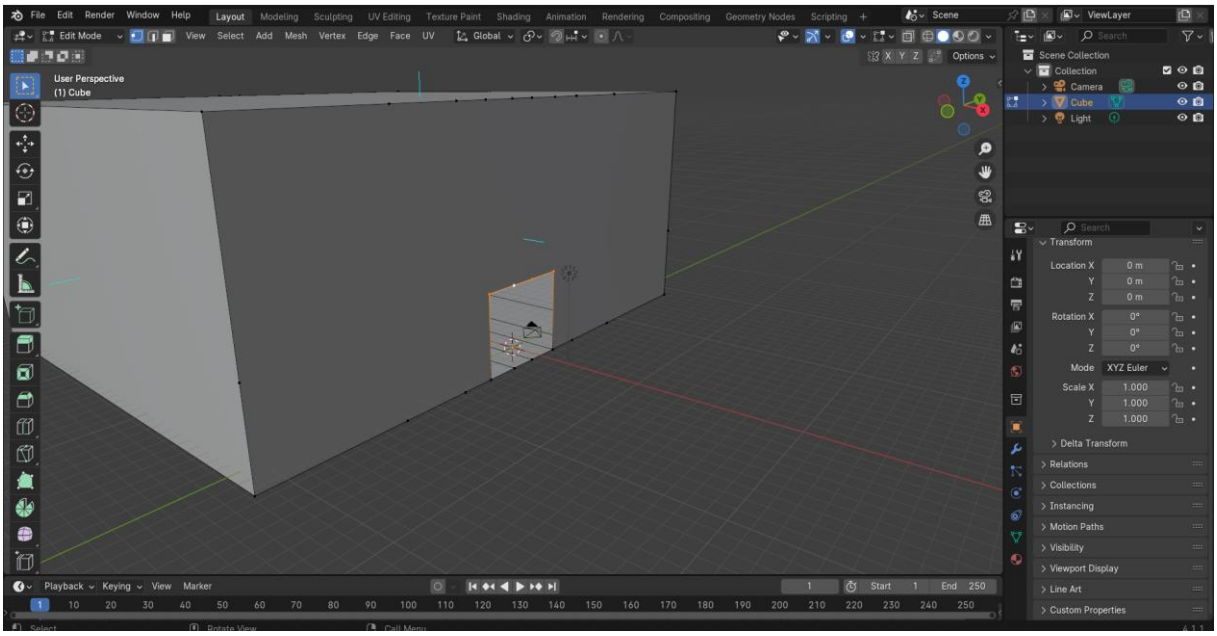
Tablica 5. Tehničke postavke projekta

Game Engine	Unity
Program za modeliranje	Blender
Programski jezik	C#
VR platforma	Oculus Quest 2
Jezik teksta u aplikaciji	Hrvatski
Jezik govora u aplikaciji	Hrvatski

13.3. Implementacija

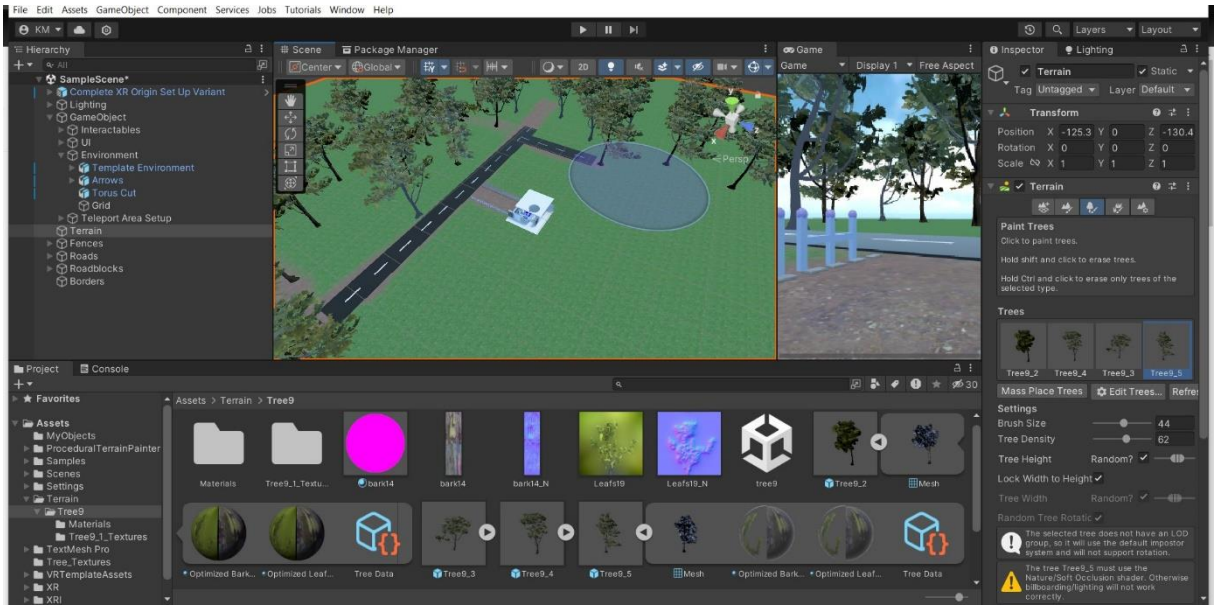
Implementacija rješenja počela je izradom jednostavnog „prototipa“ u kojemu su implementirane osnovne mehanike aplikacije. To su kontinuirano kretanje, okretaje, pritisak gumba i stavljanje elemenata u ruku.

Nakon izgradnje glavnih elemenata aplikacije, bilo je potrebno izmodelirati objekte u Blenderu. Cilj je bio izraditi prvo sve modele za početnu scenu (eng. *hub*). Početna scena bi služila kako bi se korisniku pružile upute i informacije vezane za aplikaciju. Kada se korisnik približi vratima galerije, scena bi se automatski prebacila na scenu u samoj galeriji. Na slici 17, prikazano je dizajniranje galerije u Blenderu.



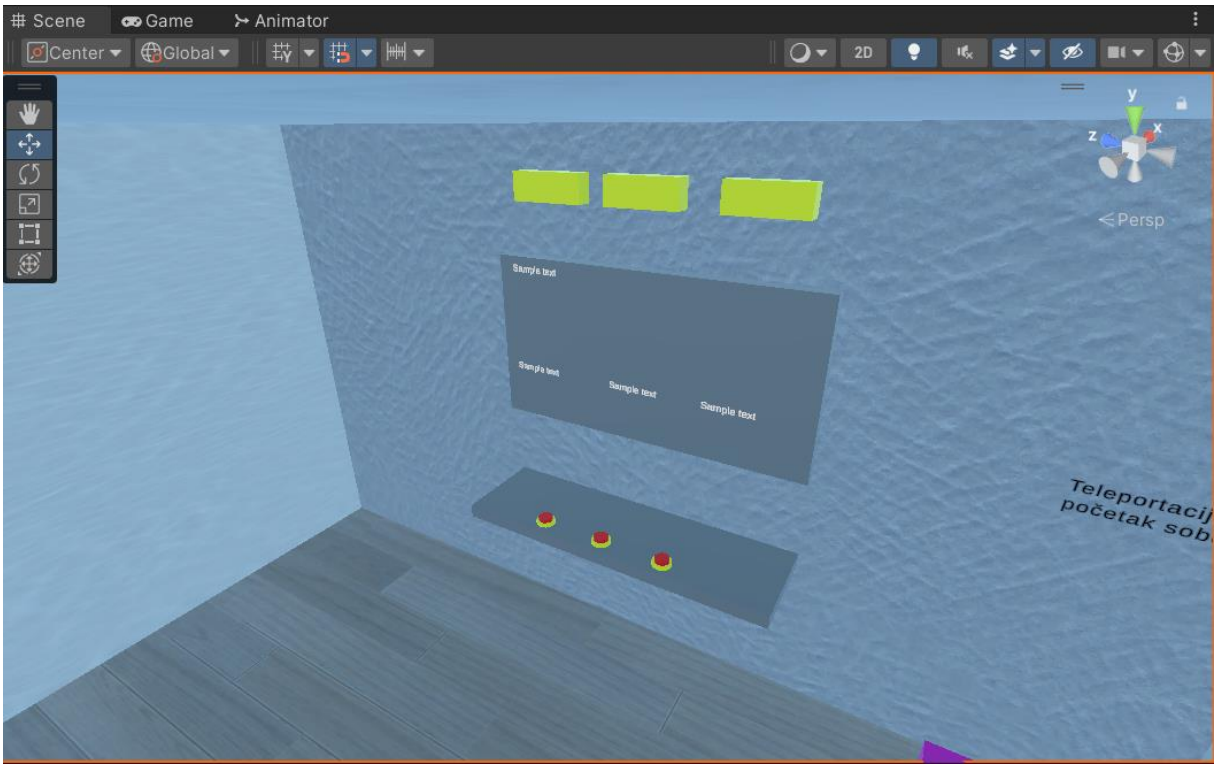
Slika 17. Dizajniranje galerije

Nakon modeliranja, u Unityju je stvoren teren, a zatim je pobojan s teksturama. Nakon toga, slijedi prijenos i postavljanje kreiranih modela u scenu. Osim vlastitih modela, korišteni su i vanjski besplatni resursi kao što su modeli drveća, a prikaz pošumljavanja scene vidljiv je na slici 18.



Slika 18. Pošumljavanje scene

Nakon što je početna scena postavljena, modelirala se je unutrašnjost galerije. Galerija se sastoji od 3 sobe; prva soba je odmah dostupna i u njoj se nalaze neke od najpoznatijih slika iz renesanse. Druge dvije sobe (impresionizam i barok) su na početku zaključane i otključavaju se tek dok se riješi kviz. Na slici 19. prikazana je stanica na kojoj se rješava kviz.



Slika 19. Stanica za rješavanje kviza

Rješavanje kviza implementirano je pomoću događaja. Korisnik bira jedan od 3 odgovora tako da pritisne gumb. Klasa *QuizManager* odgovorna je za određivanje je li pritisnut ispravan gumb. Dio koda koji provjerava odgovor je prikazan ispod:

```
public void CheckAnswer(int index) {
      if (streak == 3) return;
      int correctIndex = FindCorrectnessIndex();
      if (index != correctIndex) {
            ResetQuiz();
            onIncorrectAnswer.Invoke();
            streak = 0;
      } else {
             if (streak != 2) {
                  ChangeQuestion();
                  onCorrectAnswer.Invoke();
                  streak++;
            } else {
            onCorrectAnswer.Invoke();
            questionText.text = successMessage;
            doorsManager.openedDoors.Add(doorToOpen);
            ClearAnswers();
            if(doorToOpenLight != null) ChangeMaterial(doorToOpenLight);
            streak++;
            }
      }
}
```

Varijabla *streak* predstavlja broj uzastopnih točnih odgovora. Za uspješno rješavanje kviza, potrebno je odgovoriti 3 puta točno za redom. Odmah na početku provjerava se je li *streak* jednak 3 i ako je, tada znači da je kviz riješen i nema potrebe za provjeravanjem odgovora. Ako korisnik nije riješio kviz, potrebno je dohvatiti indeks točnog odgovora. Klasa *Question* ima 3 svojstva: pitanje, niz ponuđenih odgovora i *boolean* niz u koji je zapisano koji je odgovor točan, a koja 2 su kriva. Inače, niz ponuđenih odgovora i niz točnosti predstavljaju asocijativno polje (*Dictionary* u C#), no Unity ne može prikazati takvo polje u svojem *Inspectoru* pa je potrebno koristiti odvojena polja.

Nakon što se pronađe indeks odgovora, provjerava se je li taj indeks jednak indeksu pritisnutog gumba. Ako se indeksi ne podudaraju, kviz se restartira. Ako su jednaki, potrebno je provjeriti je li to bilo posljednje pitanje kviza. Ako nije posljednje, mijenja se pitanje i *streak* se inkrementira. Ako je bilo posljednje, potrebno je otključati vrata, prikazati odgovarajuću

poruku korisniku i pomoću aktiviranja svijetla iznad vrata pružiti korisniku uvid u to koja su vrata otključana. Otvaranje vrata pokreće se na pritisak gumba i pri tome izvršava se sljedeći kod:

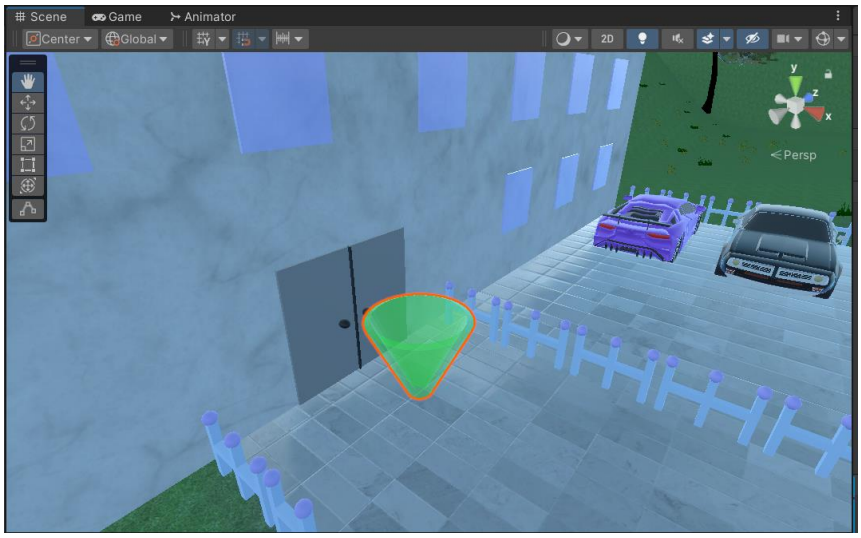
```
private IEnumerator LiftDoor() {

        wasPressed = true;
        isLifting = true;
        float elapsedTime = 0f;
        Vector3 targetPosition = initialDoorPosition + new
Vector3(0, 5f, 0);
        while (elapsedTime < 3f) {
            door.transform.localPosition                                =
Vector3.Lerp(initialDoorPosition, targetPosition, elapsedTime / 3f);
            elapsedTime += Time.deltaTime;
            yield return null;
        }
        door.transform.localPosition = targetPosition;
        isLifting = false;
}
```

Ova metoda vraća IEnumerator jer je potrebno vrijeme, točnije 3 sekunde da se vrata podignu. Tijekom 3 sekunde, lokalna pozicija objekta se mijenja pomoću metode *Lerp.* Kada se vrata otvore, korisnik može vidjeti slike novog razdoblja i riješiti novi kviz kako bi otključao novu, treću, a ujedno i posljednju sobu.

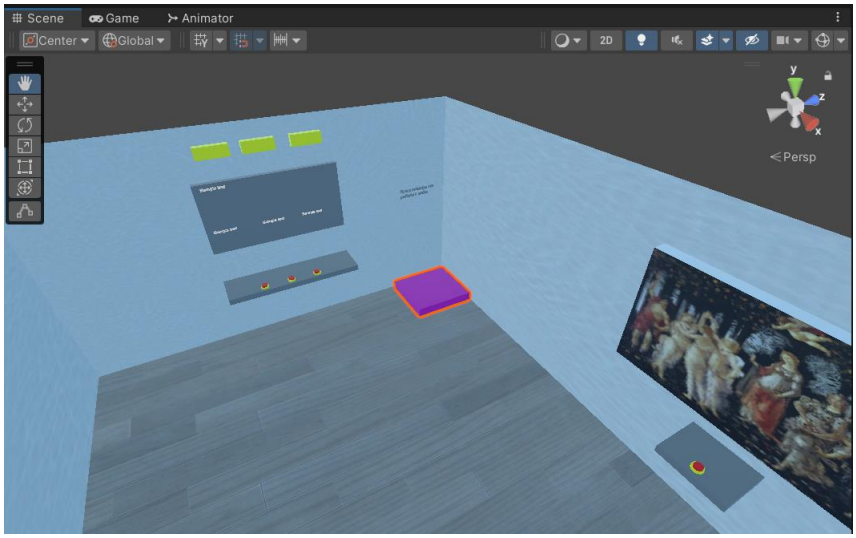
Napredak korisnika se sprema kako ne bi morao ponovno rješavati kvizove za otvaranje vrata koja je već otvorio. Za spremanje se koristi klasa *PlayerPrefs.* Vrata su obilježena pomoću id-a i posljednja otvorena vrata spremaju pomoću metode *SetInt*. Kada se aplikacija učita, otključavaju se sva vrata koja imaju jednaki ili manji id od onoga id-a koji je spremljen. Sljedeći kod izvršava otključavanje svih vrata koja je korisnik otključao:

```
private void OpenUnlockedDoors(int currentDoor) {
    for(int i = 0; i < currentDoor; i++) {
        var doorData = doors[i].GetComponent<DoorData>();
        if (doorData.id  <= currentDoor) {
            doorData.isOpen = true;
            doors[i].transform.localPosition += new Vector3(0, -1000, 0);
            Renderer renderer = lights[i].GetComponent<Renderer>();
            renderer.material = greenColor;
        }
    }
}
```

Od ostalih implementiranih značajki, treba spomenuti i teleportaciju. Korisnik ulazi i izlazi iz galerije tako što se teleportira kada uđe u objekt za teleportaciju. Logika iza teleportiranja je jednostavna; čim korisnik dodirne objekt, njegova pozicija će se postaviti na predefinirane vrijednosti. Također, na početku i na kraju svake sobe u galeriji nalaze se objekti na koje korisnik može stati kako bi se teleportirao na početak, odnosno kraj sobe. Na slici 20. prikazan je objekt koji teleportira korisnika u galeriju, a na slici 21. objekt na kojega korisnik može stati kako bi se teleportirao na početak sobe.



Slika 20. Teleportiranje u galeriju



Slika 21. Teleportiranje na početak sobe

Informacije za slike pružaju se preko snimljenog glasa koji se aktivira na pritisak gumba. Kada se pritisne gumb, skripta aktivira *AudioSource* koji započinje s radom. Ova linija koda aktivira *AudioSource:*

```
audioSource.Play();
```

13.4. Testiranje

Testiranjem su se pokušali pronaći svi problemi koji nisu uočeni tijekom implementacije. Tijekom implementacije, aplikaciju je trebalo konstantno pokretati i testirati pa veći problemi nisu uočeni. Uglavnom su tijekom testiranja pronađeni grafički problemi, a ne oni koji se tiču programske logike. Na nekoliko mjesta su pronađeni leteći i krivo postavljeni objekti. Još jedan problem koji se javio su bili objekti neproporcionalne veličine. Tako je, na primjer, stol bio manji od ograde, vrata su bila manja od igrača i slično.

Postojao je i jedan problem s animacijama. Objekt koji služi kao okidač za teleportiranje se okreće, podiže i spušta. U aplikaciji postoje 2 takva objekta; jedan se nalazi kod ulaza, a drugi kod izlaza iz galerije. Animacija koristi lokalne podatke o poziciji objekta i kada se je kopirala na drugi objekt, prilikom pokretanja igre, taj objekt bi se teleportirao na lokaciju prvog objekta. Stoga, bilo je potrebno stvoriti istu animaciju, samo s različitim podacima o poziciji za drugog objekta.

Na kraju, uvidjeli su se problemi oko zvuka. Kada bi se zvuk aktivirao, on bi se jednako čuo koliko god daleko se udaljili. Stoga, sada je sav zvuk trodimenzionalan i ima ograničeni domet. Uz to, nisu se čuli koraci korisnika pa su oni naknadno dodani.

Kod zahtjeva kvalitete nema problema. Po pitanju performansi, aplikacija radi bez ikakvih problema i na nešto starijim računalima. Ne koristi se internet, pa je korisnicima uvijek dostupna. Testirana je na operacijskom sustavu Windows 10 i na uređaju Quest 2 i radi bez problema.

13.5. Izdavanje

Tijekom faze izdavanja, izrađen je priručnik za korištenje i upute za instalaciju. Priručnik za korištenje sastoji se natuknica koji gumb radi koju stvar. Priručnik za instalaciju opisuje korake koje korisnici trebaju pratiti kako bi instalirali aplikaciju. Obje datoteke nalaze se u instalacijskoj datoteci.

13.6. Održavanje

Održavanje podrazumijeva aktivnosti naknadnog modificiranja i popravljanja softvera. Ova faza u ovome kontekstu nije relevantna jer ne postoji klijent koji bi zadao nove funkcionalnosti. Nema potrebe za modifikacijama, a niti za popravljanjem jer su se tijekom testiranja popravile sve uočene greške.

14. Prednosti i nedostaci vodopadnog modela kod izrade VR Galerije

Uočeno je mnogo pozitivnih strana vodopadnog modela za izradu aplikacije prividne stvarnosti. Stvar koja se ističe jest fokusiran, strogo definiran način izrade projekata. U svakom trenutku je jasno u kojoj fazi razvoja se projekt nalazi i što je još potrebno da se ta faza završi. Krajnji cilj izrade je konstantno jasan; kod nekih agilnih metoda, cilj se može pogubiti, no ovdje je jako teško izgubiti se.

Od svih faza, prema važnosti se ističe faza dizajna sustava. Pošto je ovaj projekt bio manjeg opsega, a vodopadni model je dobar za takve projekte, bilo je moguće u potpunosti dizajnirati sustav. Razmišljanje o sustavu i njegovo dizajniranje kroz dijagram klasa i aktivnosti puno je olakšao fazu implementacije. Iako se tijekom implementacije dijagram klasa nije vjerno pratio jer je izrazito teško predvidjeti sve na početku, svejedno je olakšao i ubrzao izradu aplikacije jer nije bilo potrebno razmišljati o sustavu tijekom implementacije.

Nedostatak dizajna sustava kod izrade aplikacije prividne stvarnosti je na koji način predočiti okruženje i UI elemente. Moguće je izrađivati 3D prototipe izgleda svijeta, no upitna je efikasnost toga jer bi utrošak vremena potencijalno bio prevelik.

Pošto se vodopadni model može koristiti za izradu bilo koje vrste aplikacija, on nema definirane faze specifične za izradu aplikacija prividne stvarnosti. Na primjer, nije definirana faza modeliranja. To je u jednu ruku dobro jer ne postoje ograničenja, već se potpuno slobodno mogu kombinirati *Unity* i *Blender* ili neki drugi softver za modeliranje 3D objekata. S druge strane, ta sloboda se može smatrati negativnom jer implementacija postaje neusmjerena. Tijekom izrade VR Galerije, konstantno se skakalo s *Unityja* u *Blender* i obratno. Potencijalno je to oduzelo više vremena nego da su se prvo izmodelirali svi objekti, a zatim da se pređe na implementaciju. Sličan „problem“ može se pronaći i za fazu dizajna sustava. Iako kasnije puno pomogne, na početku se izgubi jako puno vremena koje bi se moglo iskoristiti za raniji početak implementacije rješenja.

Kod malih aplikacija kakva je VR Galerija, neke faze razvoja nemaju preveliko značenje. Konkretno se misli na testiranje i izdavanje. Tijekom implementacije se stalno testirao program i nije ostalo mnogo stvari za popraviti. Vjerojatno bi ovdje bilo najbolje provesti beta testiranje jer bi se time uštedio i novac i vrijeme kojeg tim troši na testiranje.

Izostanak faze u kojoj se prototipira rješenje još je jedan nedostatak vodopadnog modela, pogotovo kada se radi s nekom tehnologijom s kojom programeri nemaju puno iskustva. U takvom slučaju, u fazi dizajna je potrebno nagađati što bi moglo raditi, a što ne. Na

primjer, tijekom dizajna postojala je ideja korištenja asocijativnog polja *Dictionaryja* u koji bi se spremali odgovori i njihova točnost, no tek je tijekom implementacije otkriveno da to nije moguće koristiti na željeni način.

Vjerojatno najveći problem ove metode jest njegova rigidnost koja narušava fleksibilnost i skalabilnost sustava. Bilo kakve promjene ili implementacija novih funkcionalnosti bi bila gotovo nemoguća. Da se ide nešto novo dodavati, cijela scena bi se trebala mijenjati, a vrlo vjerojatno bi se trebale raditi i promjene u skriptama.

15. Zaključak

U prvom dijelu rada bila je predstavljena prividna stvarnost. Pošto je to i dalje relativno nepoznata tehnologija, definirano je što je to prividna stvarnost, povijesni razvoj prividne stvarnosti, primjena prividne stvarnosti, tehnologija koja stoji iza prividne stvarnosti itd.

Drugi dio rada odnosi se na modele razvoja kojima se mogu izraditi prividne stvarnosti. Modeli su podijeljeni na formalne i one koje se fokusiraju samo na prividnu stvarnost. Formalni modeli dijele se na tradicionalne i agilne. U radu su opisani najvažniji tradicionalni modeli, kao što su vodopadni, inkrementalni i spiralni model, te najvažniji agilni modeli, uključujući ekstremno programiranje i Scrum. Također, navedene su i razlike između različitih modela, kao i prednosti i nedostaci svakog modela. Istim principom je pružen pregled ekstremnog programiranja i Scruma.

Treći dio rada odnosio se na izradu aplikacije prividne stvarnosti pomoću nekog od formalnih modela. Pošto je aplikacija bila manjeg opsega, koristio se vodopadni model koji je namijenjen upravo manjim projektima. Pomoću *Unityja*, izrađena je aplikacija VR Galerija u kojoj korisnici mogu pregledavati slike iz 3 različita razdoblja: renesanse, baroka i impresionizma. Galerija je *gamificirana* kroz rješavanje kvizova kako bi se otključala nova razdoblja. Cijeli razvojni proces je dokumentiran, a na kraju su pružene prednosti i nedostaci koje su ouočene tijekom rada na aplikaciji.

Na posljetku, vodopadni model se pokazao kao dobar model za izradu aplikacija prividne stvarnosti. Svakako, ovaj model nije dobar za sve projekte. Prvo se treba analizirati projekt i tek tada se mora donijeti odluka o modelu koji će se koristiti.

Popis literature

[1] S. M. LaValle, *VIRTUAL REALITY*. 2020.

[2] H. E. Lowood, „virtual reality“, *Britannica*. 2024. Pristupljeno: 29. veljača 2024. [Na internetu]. Dostupno na: https://www.britannica.com/technology/virtual-reality

[3] J. Brown, „Jaron Lanier, “the Father of VR”, addresses tech enthusiasts at UC Berkeley“. Pristupljeno: 12. ožujak 2024. [Na internetu]. Dostupno na: https://cio.ucop.edu/jaron-lanier-the-father-of-vr-addresses-tech-enthusiasts-at-uc-berkeley/

[4] VR.Space, „ Jaron Lanier, Father of VR“. Pristupljeno: 22. ožujak 2024. [Na internetu]. Dostupno na: https://vr.space/news/education/jaron-lanier/

[5] J. Lanier, *Dawn of the New Everything*, 1. izd. New York: Henry Holt and Company, 2017.

[6] I. E. Sutherland, „The Ultimate Display“, 1965. Pristupljeno: 22. ožujak 2024. [Na internetu]. Dostupno na: https://worrydream.com/refs/Sutherland_1965_-_The_Ultimate_Display.pdf

[7] T. Mazuryk i M. Gervautz, „Virtual Reality History, Applications, Technology and Future“, Beč, pros. 1999. Pristupljeno: 22. travanj 2024. [Na internetu]. Dostupno na: https://www.researchgate.net/publication/2617390_Virtual_Reality_-_History_Applications_Technology_and_Future

[8] D. Barnard, „History of VR – Timeline of Events and Tech Development“. Pristupljeno: 22. travanj 2024. [Na internetu]. Dostupno na: https://virtualspeech.com/blog/history-of-vr

[9] J. Glover i J. Linowes, *Complete Virtual Reality and Augmented Reality Development with Unity*. Birmingham: Packt Publishing Ltd., 2019.

[10] O. Özkan, „The Compatibility of Widely Used Presence Questionnaires with Current Virtual Reality Technology“, Bahcesehir University, Instanbul, 2016. doi: 10.13140/RG.2.2.28935.16804.

[11] T. Parisi, *Learning Virtual Reality*, 1. izd. Sebastopol: O’Reilly Media, 2015.

[12] C. Hillmann, *Unreal for Mobile and Standalone VR*, 1. izd. Berkeley, CA: Apress, 2019. doi: 10.1007/978-1-4842-4360-2.

[13] P. Jones, T. Osborne, C. Sullivan-Drage, N. Keen, i E. Gadsby, *Virtual Reality Methods*. Policy Press, 2022. doi: 10.2307/j.ctv2n7j137.

[14] S. Luo, „VR Rome“. Steven Luo, 05. studeni 2018. Pristupljeno: 18. svibanj 2024. [Na internetu]. Dostupno na: https://store.steampowered.com/app/964460/VR_Rome/

[15] „IMU“, XinReality. Pristupljeno: 19. svibanj 2024. [Na internetu]. Dostupno na: https://xinreality.com/wiki/IMU

[16] „Inside-out tracking“, *XinReality*. 2020. Pristupljeno: 01. srpanj 2024. [Na internetu]. Dostupno na: https://xinreality.com/wiki/Inside-out_tracking

[17] „Outside-in tracking“, *XinReality*. 2018. Pristupljeno: 01. srpanj 2024. [Na internetu]. Dostupno na: https://xinreality.com/wiki/Outside-in_tracking

[18] J. P. M. Masso, A. S. García, V. López-Jaquero, i P. González, „Developing VR applications: the TRES-D methodology“, Instituto de Investigación en Informática de Albacete, Albacete, 2005. Pristupljeno: 01. srpanj 2024. [Na internetu]. Dostupno na: https://www.researchgate.net/publication/249903814_Developing_VR_applications_the_TRES-D_methodology

[19] U. S. Senarath, „Waterfall Methodology, Prototyping and Agile Development“, lip. 2021. Pristupljeno: 02. srpanj 2024. [Na internetu]. Dostupno na: https://www.researchgate.net/publication/353324450_Waterfall_Methodology_Prototyping_and_Agile_Development

[20] S. M. Ali Khan, „Waterfall Model Used in Software Development Reference: Software Requirements Engineering Waterfall Model“, lip. 2023. Pristupljeno: 02. srpanj 2024. [Na internetu]. Dostupno na: https://www.researchgate.net/publication/371902449_Waterfall_Model_Used_in_Software_Development_Reference_Software_Requirements_Engineering_Waterfall_Model

[21] K. Petersen, C. Wohlin, i D. Baca, „The Waterfall Model in Large-Scale Development“, 2009, str. 386–400. doi: 10.1007/978-3-642-02152-7_29.

[22] S. Atanasijević, „ŽIVOTNI CIKLUS RAZVOJA SOFTVERA“, srp. 2022. Pristupljeno: 02. srpanj 2024. [Na internetu]. Dostupno na: https://www.researchgate.net/publication/362001817_ZIVOTNI_CIKLUS_RAZVOJA_SOFTVERA

[23] S. M. Ali Khan, „V-Model Used in Software Development“, lip. 2023. Pristupljeno: 02. srpanj 2024. [Na internetu]. Dostupno na: https://www.researchgate.net/publication/371902849_V-Model_Used_in_Software_Development

[24] N. M. Ali Munassar i A. Govardhan, „A Comparison Between Five Models Of Software Engineering“, *International Journal of Computer Science Issues*, lis. 2010, Pristupljeno: 02. srpanj 2024. [Na internetu]. Dostupno na: https://www.researchgate.net/publication/344932403_A_Comparison_Between_Five_Models_Of_software_Engineering

[25] M. Stoica, M. Mircea, i B. Ghilic-Micu, „Software Development: Agile vs. Traditional“, *Informatica Economica*, sv. 17, izd. 4/2013, str. 64–76, pros. 2013, doi: 10.12948/issn14531305/17.4.2013.06.

[26] R. Pressman, *Software Engineering: A Practitioner's Approach*, 5. izd. New York: The McGraw-Hill Companies, Inc., 2000.

[27] G. Caldwell, *Agile Project Management: The Complete Guide for Beginners to Scrum, Agile Project Management, and Software Development*. Greg Caldwell , 2019.

[28] A. Shrivastava, I. Jaggi, N. Katoch, D. Gupta, i S. Gupta, „A Systematic Review on Extreme Programming“, *J Phys Conf Ser*, sv. 1969, izd. 1, str. 012046, srp. 2021, doi: 10.1088/1742-6596/1969/1/012046.

[29] I. Purnama, „Clinical Information System Using Extreme Programming Method“, *International Journal of Science, Technology & Management*, sv. 4, izd. 5, str. 1229–1235, ruj. 2023, doi: 10.46729/ijstm.v4i5.931.

[30] W. Zayat i O. Senvar, „Framework Study for Agile Software Development Via Scrum and Kanban“, *International Journal of Innovation and Technology Management*, sv. 17, izd. 04, lip. 2020, doi: 10.1142/S0219877020300025.

[31] V. Hema, S. Thota, S. Naresh Kumar, C. Padmaja, C. B. Rama Krishna, i K. Mahender, „Scrum: An Effective Software Development Agile Tool“, *IOP Conf Ser Mater Sci Eng*, sv. 981, izd. 2, pros. 2020, doi: 10.1088/1757-899X/981/2/022060.

[32] P. Adi, „Scrum Method Implementation in a Software Development Project Management“, *International Journal of Advanced Computer Science and Applications*, sv. 6, izd. 9, 2015, doi: 10.14569/IJACSA.2015.060927.

[33] H. Majeed, „Issues and Challenges In Scrum Implementation“, *Int J Sci Eng Res*, 2012.

[34] J. P. Molina Masso, A. S. García, V. López-Jaquero, i P. González, „Developing VR applications: the TRES-D methodology“, lis. 2005. Pristupljeno: 06. srpanj 2024. [Na internetu]. Dostupno na: https://www.researchgate.net/publication/249903814_Developing_VR_applications_the_TRES-D_methodology#fullTextFileContent

[35] A. Paszkiewicz, M. Salach, P. Dymora, M. Bolanowski, G. Budzik, i P. Kubiak, „Methodology of Implementing Virtual Reality in Education for Industry 4.0“, *Sustainability*, sv. 13, izd. 9, tra. 2021, doi: 10.3390/su13095049.

[36] J. Polcar, M. Gregor, P. Horejsi, i P. Kopecek, „Methodology for Designing Virtual Reality Applications“, 2016, str. 768–774. doi: 10.2507/26th.daaam.proceedings.107.

[37] N. Dozio *i ostali*, „A design methodology for affective Virtual Reality“, *Int J Hum Comput Stud*, sv. 162, lip. 2022, doi: 10.1016/j.ijhcs.2022.102791.

Popis slika

Popis tablica