Projektni zadatak

-Home Security System-

PREDMET: SE311-Arhitektura i projektovanje softvera
PREDMET: SE321-Obezbeđenje kvaliteta, testiranje i održavanje softvera

Asistent:
MSc Nebojša Gavrilović

Student:
Predrag Jevtić
Br. Indeksa: 3768

Beograd, 2021.

SADRŽAJ:

Beograd, 2021.

1. Uvod

Ovim dokumentom predstavlja se arhitektura HomeSecurity sistema . Takođe, nakon pregleda arhitekture sistema, biće istestirane određene funkcije i biće prikazano da li jedan ovakav sistem zaista i zadovoljava potrebe onoga ko se odluči za njegovo korišćenje.

Deo dokumenta koji se odnosi na arhitekturu sistema, sadrži sledeće elemente:

- Početnu reč o samom projektu kao i korišćene metodologije u izradi arhitekture
- Predstavljanje slučajeva korišćenja našeg sistema UseCase dijagramima
- Način konstrukcije našeg sistema na dva tipa arhitekture
- Zahmanovu matricu
- “4+1” pogled implementacije sistema

Svaka stavka biće propraćena odgovarajućim tekstualnim objašnjenjem, kao i odgovarajućim tipom dijagrama napravljenog u PowerDesigner-u.

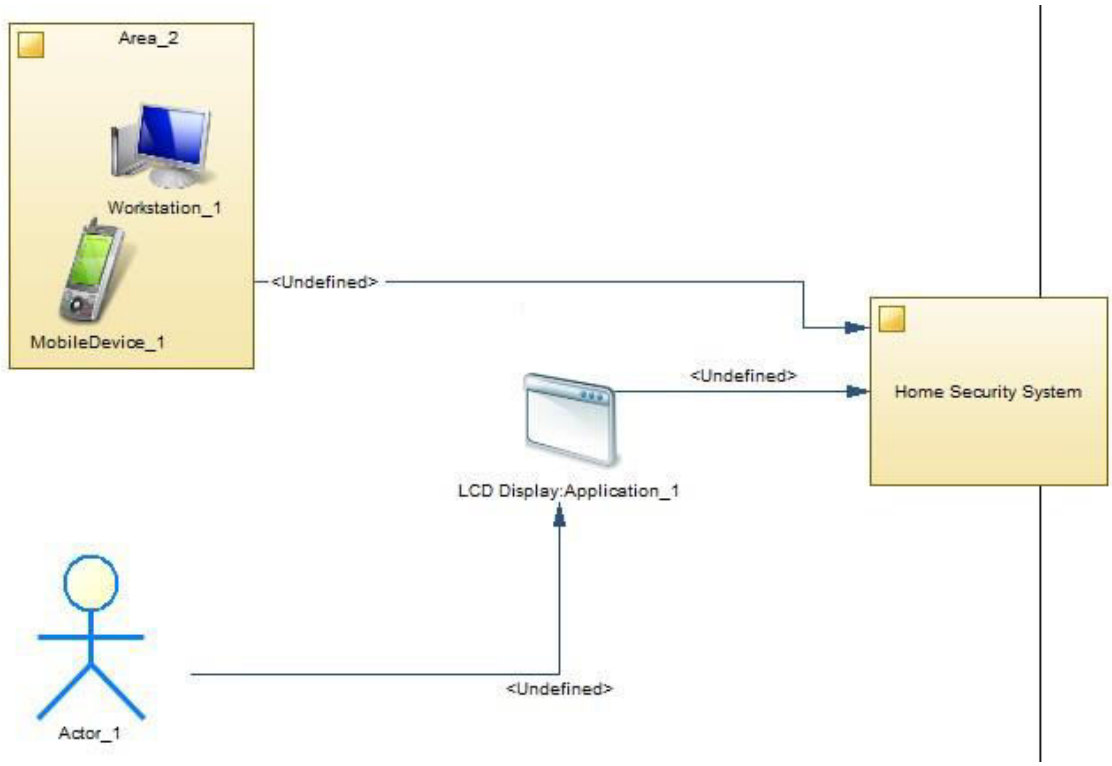
Nakon dela o arhitekturi, sledi deo dokumenta koji se odnosi na testiranje koji sadrži sledeće elemente:

- Master plan testiranja, koji se sastoji iz delova kao što su strategija, stavke koje će biti testiranje, stavke koje neće biti testirane, pass/fail testiranje itd.
- Metriku softvera

Nakon finalizacije svih tačaka, dokument će biti spreman za dalju upotrebu i predstavljaće dobru osnovu za samu implementaciju sistema.

2. O projektu

Sistem za zaštitu domova je sistem koji je zamišljen tako da putem senzora otkriva eventualne probleme, a zatim se oglašava alarmom. Senzori su raspoređeni svuda po kući ili stanu i detektuju pokrete,dim i toplotu, iz čega zaključujemo da će se sistem oglašavati ukoliko dođe do provale u kuću ili ukoliko dođe do požara. Takođe, postoje i takozvani magnetni senzori koji se ugrađuju u vrata i prozore i detektuju svako otvaranje istih ukoliko je mod za magnetne senzore uključen. Sistemu se pristupa direktno putem uređaja koji može da bude instaliran bilo gde u domu (obično je to kod ulaznih vrata da bi se sistem uključivao prilikom izlaska iz doma) ili indirektno, putem mobilnog telefona ili lap topa koji su povezani sa glavnim uređajem. Postoji i još jedan uređaj koji se nalazi izvan kuće, takođe obično kod ulaznih vrata, i on isključivo služi tome da bi se sistem deaktivirao prilikom našeg ulaska u kuću. U oba slučaja, korisnik pristupa putem svoje lozinke. Sistem poseduje i kamere raspoređene van kuće, čije je snimanje moguće videti na glavnom displeju, ili nekom od povezanih uređaja. Sam sistem moguće je ručno isprogramirati i napraviti svoj program zaštite doma. Ukoliko je sa sistem povezan neki od dodatnih uređaja (npr. moblini telefon) , moguće je podesiti da na njemu dobijemo obaveštenje ukoliko dođe da aktivacije alarma. U daljem tekstu svi gore navedeni problemi biće dataljno razloženi i objašnjeni.



Sistemu je moguće pristupiti direktno preko LCD displeja sistema, ili putem uređaja povezanih sa njim

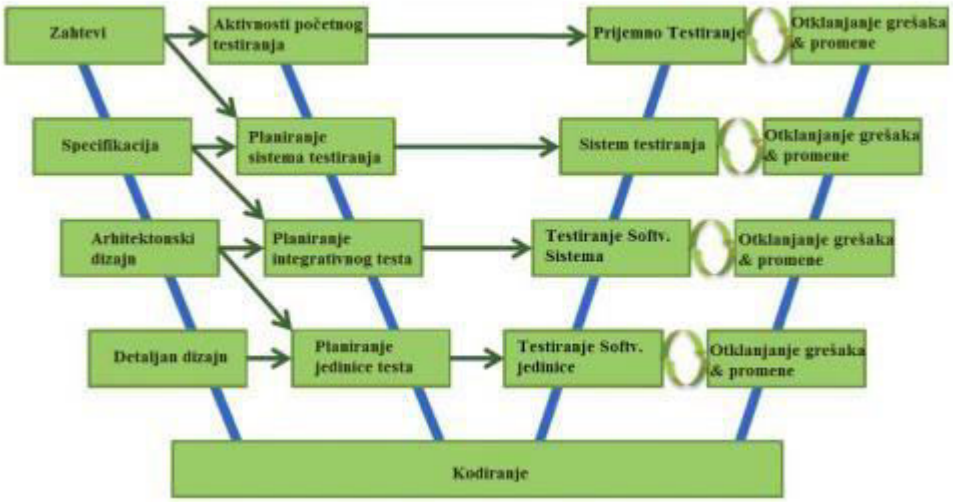
2.1 Projektni tim

Projektni tim se sastoji iz:

- Menažera projekta – on ima centralnu poziciju u projektu. Zadužen je za donošenje bitnih i krucijalnih odluka koje se odnose na sve aspekte realizacije projekta, ali i svih aktivnosti na projektu kao što su trošenje resursa i finansijskih sredstava, koordinacija i usmeravanje članova ostalih timova. Pored toga zadužen je i za celokupnu komunikaciju vezanu za projekat. Takođe poželjno je da pozitivno utiče na sve članove svog tima kako bi njihov rad učinio efikasnijim, a atmosferu manje stresnom.
- Razvojnog tima – zaduženi su za davanje predloga o potencijalnim arhitektonskim i dizajn šablonima menadžeru projekta. Takođe zaduženi su za implementaciju softvera uz pomoć platforme, framework-a i programskog jezika koji su odabrani na sastancima sa menadžerom projekta.
- Tima za obuku – za ovaj tim je poželjno da se sastoji delom od članova razvojnog tima budući da oni najbolje znaju sistem koji su projektovali. Zadatak ovog tima jeste da izvrši obuku zaposlenih kako bi što bolje i efikasnije koristili sistem. Takođe ovaj tim mora biti prisutan tokom celokupnog razvoja sistema kako bi korisnicima što bolje opisali sistem i objasnili način rada.
- Tima testera – zadužen za testiranje, veoma krucijalnu fazu projekta bez koje se ne može nastaviti razvoj.
- Buisness analitičar – zadužen je za rad sa krajnjim korisnicima kako bi odredio koji su njihovi zahtevi i šta ustvari oni očekuju o sistema. Takođe on balansira odnos korisnika sa sistemom, tj. pomaže korisnicima da shvate da li je ono što traže zaista i potrebno. Zbog toga mora imati određena iskustva u softverskom inženjerstvu, testiranju, itd.

2.2 Metodologija

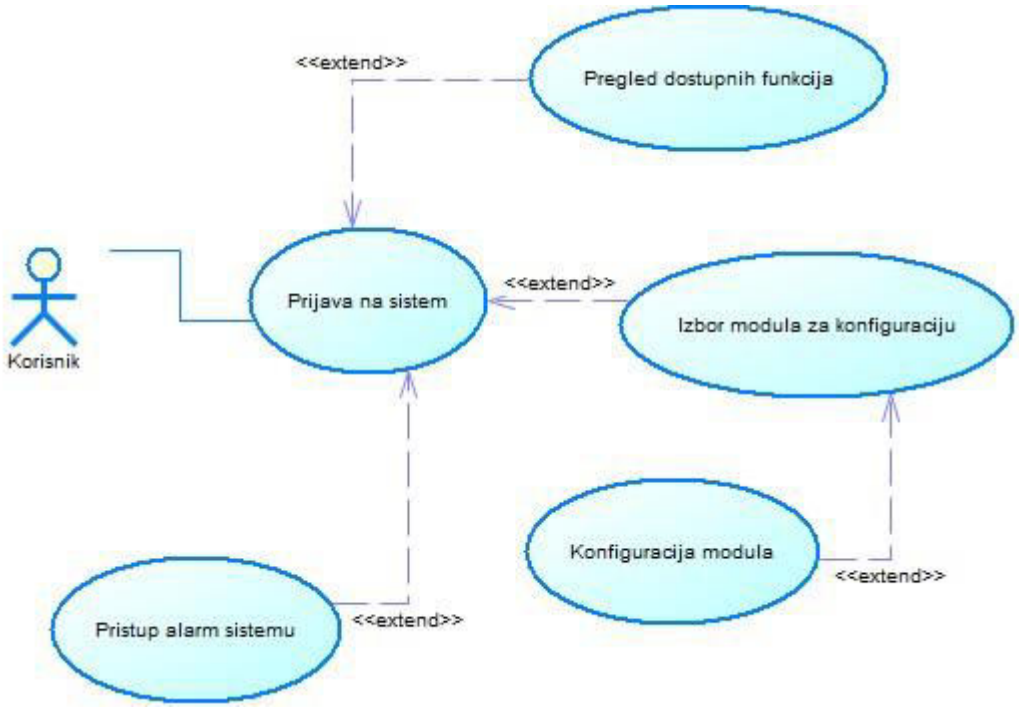
Za razvoj ovog sistema koristiće se W-model koji predstavlja modifikaciju uobičajenog Vmodela. Test proces obično dobija premalo pažnje u drugim modelima i obično se pojavljuje kao neprivlačan zadatak da se izvrši posle kodiranja. Da bi se testiranje vršilo na ravnopravnoj osnovi, drugo V je posvećeno testiranju koje se integriše u modelu. Oba V tako zajedno daju jedan kompletan i veoma pogodan model za razvoj – W model.



sl. W-model razvoja softvera

3. Slučajevi korišćenja

Pre pristupa samoj arhitekutri sistema, potrebno je dati detaljan uvid u slučajeve korišćenja, odnosno koje sve funkcije sistem treba da obezbedi korisniku i na koji način korisnik sve može koristiti sistem. U nastavku će biti predstavljen UseCase dijagram.



sl. UseCase dijagram

Sledećom tabelom, detaljnije su objašnjene funkcije samog sistema:

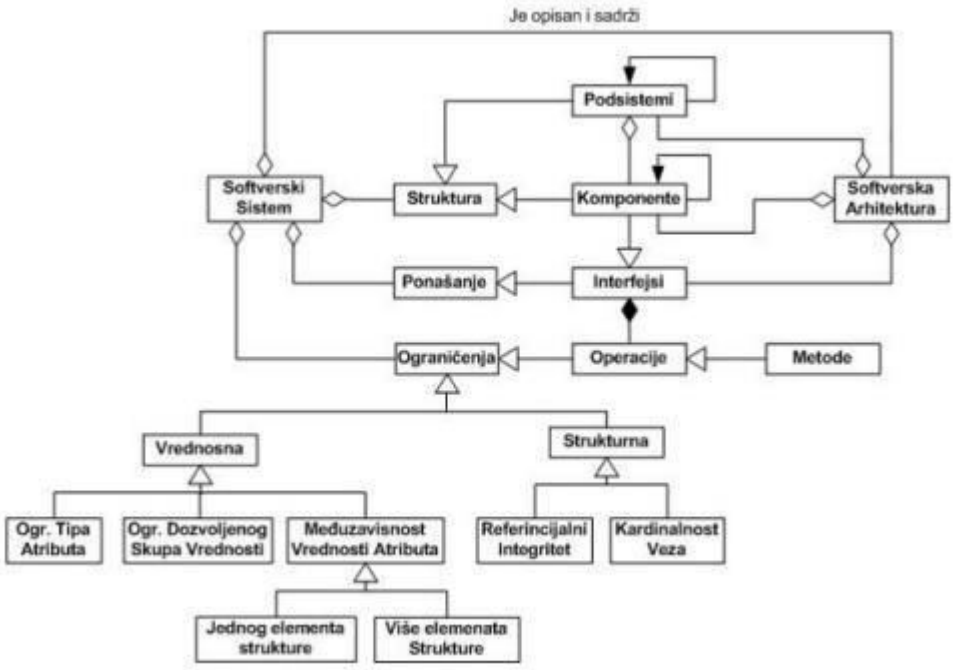
R.B	Zahtev	Opis
1	Obezbediti autorizovan pristup sistemu	Bilo da korisnik pristupa sistemu putem povezanih uređaja, ili direktno preko displey-a, sistem mora da mu obezbedi unos lozinke i njenu proveru ispravnosti
2	Obezbediti korisniku pregled dostupnih funkcija	Tek nakon prihvatanja lozinke unete od strane korisnika, obezbediti mu ispis dostupnih opcija, koje se odnosno na samu konfiguraciju sistema i upravljanja modulima
3	Obezbediti korisniku izbor određenog modula	Korisniku na jasan način treba da bude predstavljen spisak dostupnih modula, odnosno pristup svakom podsistemu pojedinačno i dalje njegovoj konfiguraciji
3.1	Obezbediti korisniku pristup	Korisniku treba da bude omogućen rad sa senzorima.

	senzor mod-u	Ovakav vid konfiguracije, povlači 4 bitna zahteva, odnosno pristup svakom tipu senzora pojedinačno: • Obezbediti korisniku opciju uključivanja i isključivanja senzora za dim, zavisno od njihovih rasporeda po sobama • Obezbediti korisniku opciju uključivanja i isključivanja senzora za pokret, zavisno od njihovih rasporeda po sobama. Senzori trebaju da prepozanaju kretanje kućnih ljubimaca i ne reaguju na takav vid kretanja • Obezbediti korisniku opciju uključivanja i isključivanja senzora za temperaturu, zavisno od njihovih rasporeda po sobama. Omogućiti podešavanje na koju temepraturu reagovati • Obezbediti korisniku opciju uključivanja i isključivanja magnetnih senzora, zavisno od njihovo rasporeda na vratima i prozorima.
3.2	Obezbediti korisniku pristup mod-u za nadzor	Korisniku treba da bude omogućen snimaka kamere, kao i live prikaz, odnosno pregled onoga što kamera trenutno “vidi” , kao i prikaz tačno određene kamere (postoji više kamera u okviru sistema nadzora)
4	Obezbediti korisniku pristup alarm sistemu	Korisniku treba da bude omogućeno da određuje tip alarma, zavisno od detektovane opasnosti

Korisnik takođe ima mogućnost pristupa sistemu putem drugih uređaja (mobilni, lap top itd), gde se vrši povezivanje putem Wi-Fi signala. U daljem radu ova opcija će samo biti uzeta u obzir, ali se neće ulaziti u njeno detaljnije predstavljanje jer bi za tako nešto bilo potrebno praviti i posebnu konstrukciju za aplikacuju koja bi u tom slučaju trebalo da bude instalirana na dodatnom uređaju.

4. Arhitektura sistema

Softverska arhitektura programa ili računarskog sistema je struktura sistema, koja uključuje softverske komponente, eksterno vidljive osobine ovih komponenti i relacije između njih. Sistem je kolekcija povezanih jedinica koje su organizovane kako bi ispunile specifičnu svrhu. Sistem može biti opisan pomoću jednog ili više modela, po mogućnosti iz različitih perspektiva. Eksterno vidljive osobine komponenti mogu biti korišćene od strane drugih komponenti. Softverska arhitektura predstavlja sam problem i domen rešenja. Ona treba da omogući efektivno zadovoljenje arhitekturalno značajnih eksplicitnih funkcionalnih zahteva, zahteva za kvalitetom i implicitnih zahteva proizvodnih linija. Konkretno za naš sistem zaštite možemo primeniti dosta tipa arhitektura, ali ovde ćemo se konkretno opredeliti za klijentserver kao i CORBA arhitekturu.



sl. Struktura softverske arhitekture

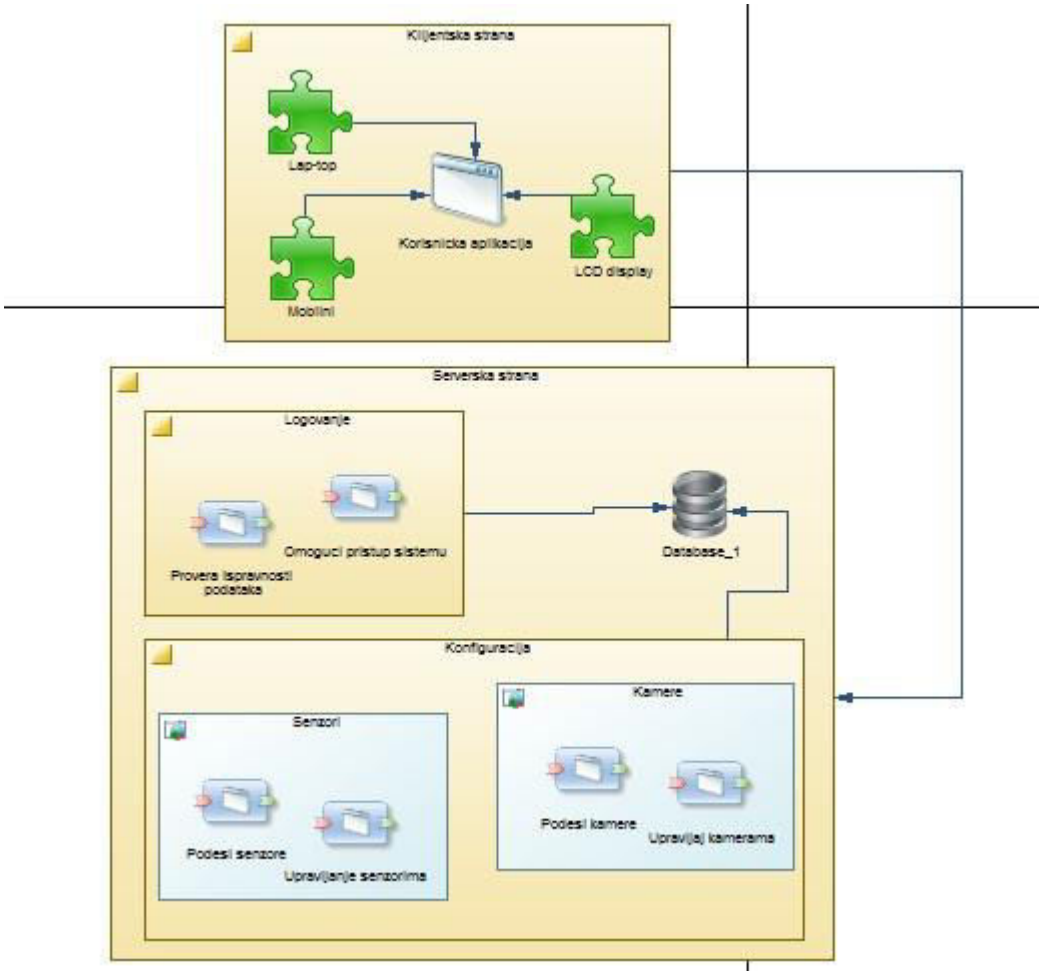
4.1 Klijent-server arhitektura

Klijent-server model je baziran na distribuciji funkcija između dva tipa nezavisnih i autonomnih procesa, to su server i klijent. Klijent predstavlja bilo koji proces koji zahteva specifične usluge od server procesa, koji obezbeđuje zahtevane usluge za određenog klijenta. U većini slučajeva su klijenti obično aktivni korisnici koji šalju zahteve i čekaju da se oni ispune, dok je server pasivan, tj. čeka zahteve, ispunjava ih i šalje natrag korisniku.

Komunikacija između klijenta i servera se obavlja preko komunukacionih kanala koji zavise od mašina na kojima se server i klijent nalaze.

sl. Struktura tipične klijent-server arhitekture

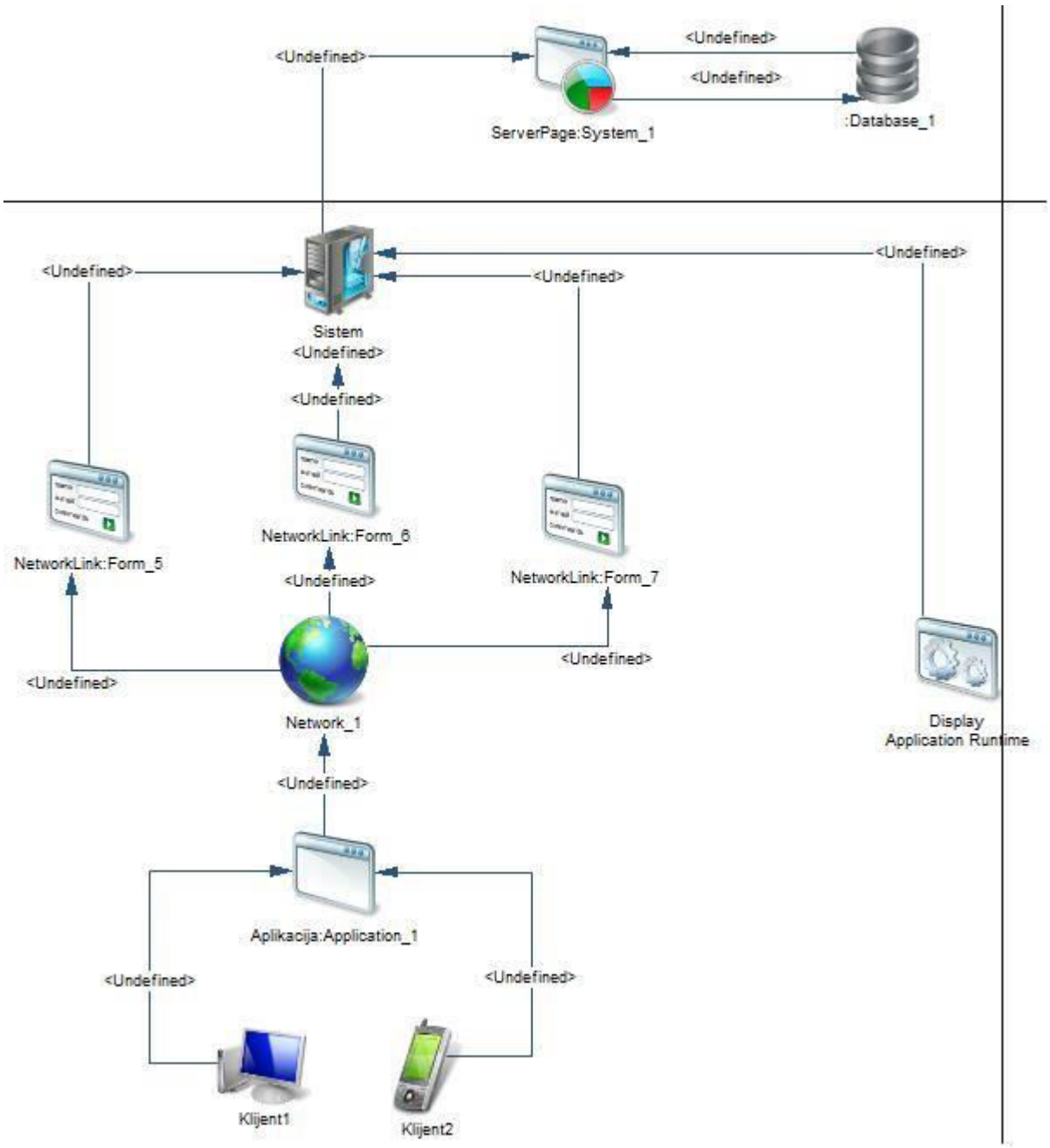
Tipična klijent server-arihtektura daje nam prikaz stanja sistema na ekranu određenog uređaja i funkcije koje su nam na raspolaganju, a u pozadini se nalaze servisi koji izvršavaju zadate zadatke.



sl. Prikaz arhitekture Home security sistema

Kao što vidimo, naš sistem se sastoji iz klijentske i server strane. U klijentskom delu imamo korisničku aplikaciju koja može da bude prikazana na više uređaja i koja se sastoji iz određenih funkcija koje želimo da zadamo samom sistemu. Sve zadatke koje zadajemo sistemu, izvršava serverska strana, odnosno servisi zaduženi za izvršavanje zadatih funkcija. Serverska strana je u startu podeljena na dva posebna podservera, koji sa jedne strane upravljaju procesom logovanja, a sa druge samom konfiguracijom sistema. Oba ova podservera čuvaju podatke u integrisanoj bazi podataka. Sam proces konfiguracije, sastoji se iz dva posebna podsistema, odnosno sistema koji se odnosi na rad sa senzorima i sistema koji se odnosi na rad sa kamerama. Tu su uključeni servisi koji omogućavaju rad i podešavanja sa senzorima i kamerama.

4.1.1 Dijagram infrastrukture



sl. Infrastruktura sistema

Sa dijagrama primećujemo da se pristup sistemu preko lap topa, ili mobilnog telefona omogućuje aplikacijom koja mora da bude instalirana na njih. Druga bitna stvar je to da je za dalji pristup sistemu potrebno da postoji internet konekcija da bi doslo do realizacije servisa. Ukoliko sistemu pristupamo direktno putem display-a nije nam potrebna internet konekcija, već se servisima pristupa direktno.

4.2 CORBA arhitektura

CORBA omogućava klijentu da poziva metode sa objekta na serveru uz nezavisnost programa, jezika i lokacije. Interakcija je omogućena preko ORB - Object Request Brokers komponenata na klijentu i na serveru (koji poseduju mehanizam RPC) a komunikacija se odvija preko IIOP (Internet Inter-ORB Protocol). Mogućnosti CORBA definisane su IDL-Interface Definition Language. Ovaj standard definiše interfejs koji bi trebalo da omogući da komponente napisane u različitim jezicima međusobno sarađuju. To je ostvareno definisanjem seta metoda koje su vidljive za ostale komponente koje mogu da budu na različitim čvorovima u mreži. CORBA model u potpunosti podržava ideju troslojne arhitekture, koja po prirodi ima aplikaciju distribuiranu između klijenta, aplikativnog servera i servera baze podataka.

Elementi CORBA standarda su:

- Object Request Broker

Common Object Request Broker Architecture specifikuje osnovne servise za komunikaciju objekata, a bazira se na mehanizmu RPC-a. ORB konstituiše arhitektonske komunikacione komponente i ponekad se naziva kao magistrala objekta ili jednostavno distributer poruka. Zadatak ORB je da uspostavi klijent server vezu između objekata, gde je klijent objekat koji poziva operaciju (funkciju) drugog objekta.

- Object Adapter

Tehnički gledano, object adapter predstavlja vezu između ORB-a i odgovarajuće implementacije objekta. Object adapter povezuje CORBA objekte koji su specificirani pomoću IDL-a (Interface Definition Language). Kada ORB primi zahtev klijenta on rutira ovaj zahtev do ORB-a na serverskoj strani. Zatim, ORB na serverskoj strani prenosi zahtev do object adaptera. Object adapter zatim treba da odredi prema kojoj implementaciji objekta treba da se prosledi zahtev.

- Interface Definition Language

Za razvoj fleksibilnih distribuiranih aplikacija, naročito za heterogene platforme, potrebno je striktno razdvajanje između intrefejsa i implementacije objekata. Objekti su definisani u posebnom programskom jeziku kao što je Interface Definition Language. IDL se koristi da definiše interfejse objektima. Za specifikaciju interfejsa objekata, važni su samo atributi i operacije kojima se može pristupiti izvan objekata – slično public deklaracijama u Java i C++ klasama.

- Repozitorijum Interfejsa

Repozitorijum Interfejsa - IR ima zadatak da smešta odnosno da čuva definicije interfejsa

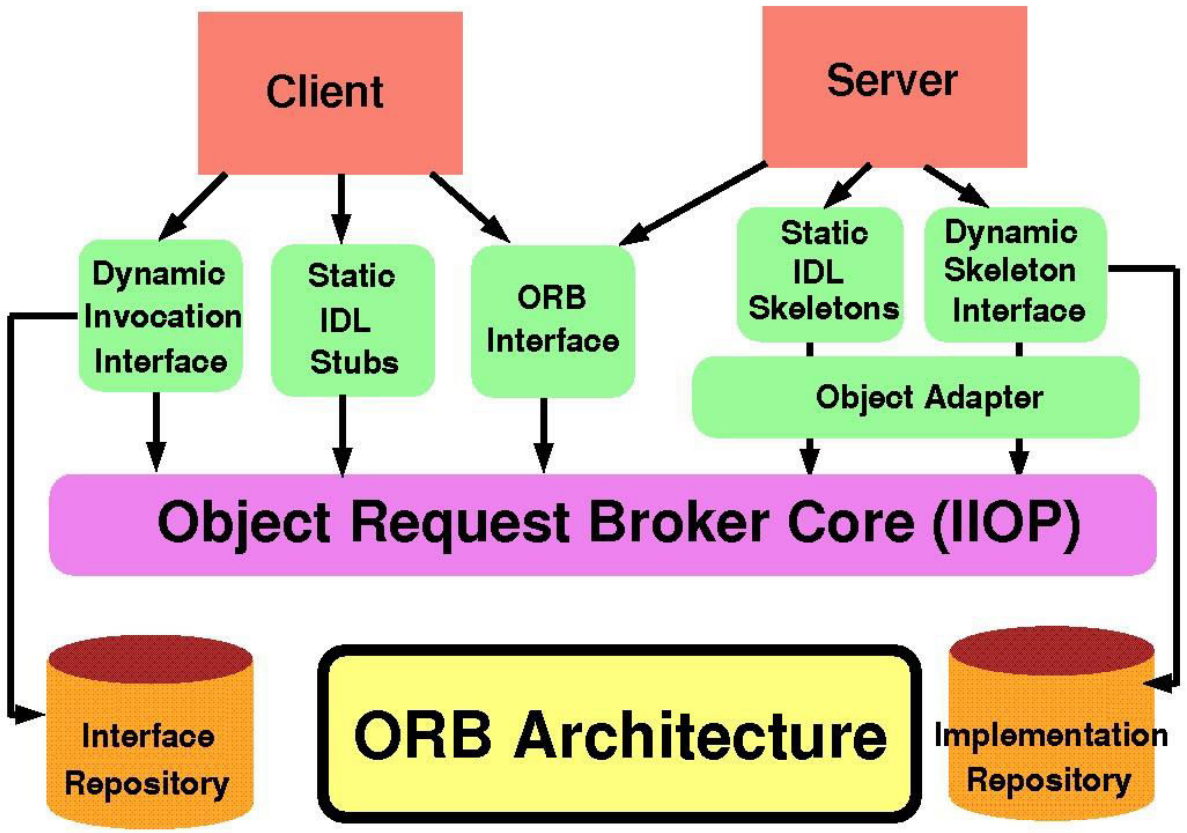
CORBA objekata. IR sam po sebi predstavlja CORBA server koji treba da bude aktiviran odvojeno od klijentskog i serverskog procesa i može biti smešten bilo gde u mreži. IR je opisan skupom IDL interfejsa određujući tako način kako se IR-u može pristupiti udaljeno.

- Dinamički Poziv Interfejsa

Postoje dva načina da klijent pozove operaciju na serveru a to su: statički ili dinamički. U oba slučaja, klijent treba da zna referencu odgovarajućeg CORBA objekta.

- Interfejs Dinamičkog Skeletona

Interfejs Dinamičkog Skeletona obezbeđuje mehanizam rada za upravljanje komponentama na serverskoj strani čija IDL definicija nije poznata za vreme implementacije i kompilacije servera.



sl. CORBA arhitektura

4.2.1 Zašto CORBA arhitektura?

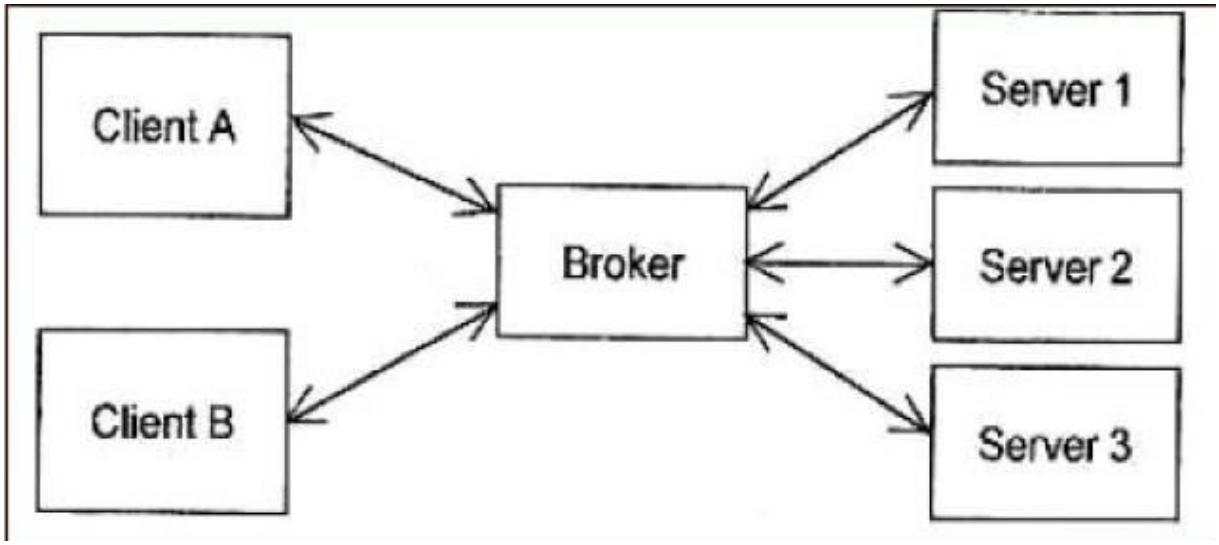
Rizici u primeni klijent – server arhitekture se ogledaju u karakteristikama kvaliteta dizajna softvera kao što su:

- Sigurnost - Sigurnost je veliki problem bez savršenog rešenja. Potrebno je upotrebiti enkripciju, zaštitne zidove (firewalls) itd.

- Potreba za adaptivnim održavanjem - Budući da se programska podrška za klijente i servere oblikuje odvojeno potrebno je osigurati da sva programska podrška bude kompatibilna prema unatrag (backwards) i prema unapred (forwards), te kompatibilna s drugim verzijama klijenata i servera.

Iz tog razloga možemo koristiti CORBA arhitekturu koja nam pruža određena poboljšanja u odnosu na klijent-server arhitekturu.

To je generalizacija klijent-server arhitekture. Znatno bolje promoviše skalabilnost i mogućnost jednostavnije modifikacije. Nastoji otkloniti neke nedostatke klasične klijentserver arhitekture, a posebno nastoji povećati performanse, raspoloživost, i sigurnost. Uopšteno govoreći tronivojska CORBA arhitektura sadrži korisnički nivo, logički ili poslovni sloj, kao i sloj podataka, odnosno serversku stranu.



sl. Način funkcionisanja troslojne arhitekture

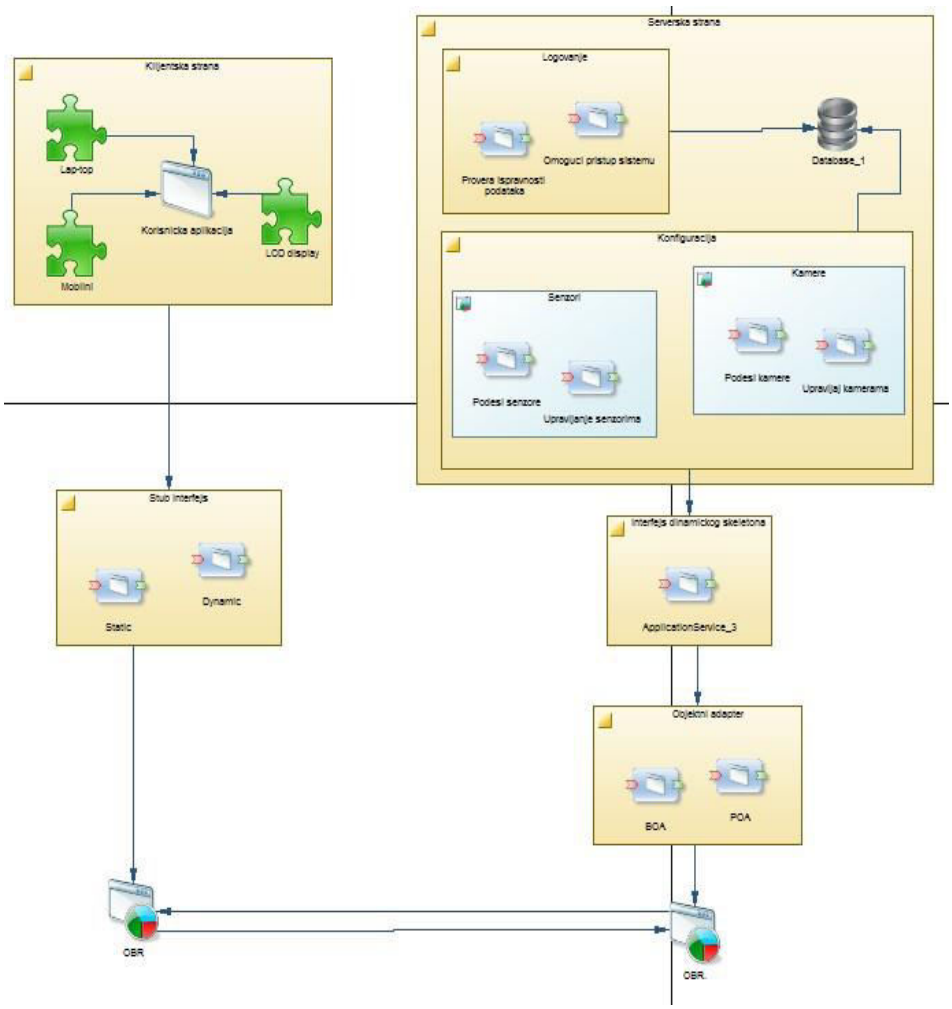
Sledeća tabela daje nam prednosti i mane ovakve arhitekture:

RB	Prednosti	Mane
1	Svaki sloj deluje kao koherentna celina	Smanjene performanse jer gornji slojevi samo indirektno dohvataju donje slojeve.
2	Vrlo jednostavna zamena sloja novijom verzijom	Teško se pronalazi ispravna apstrakcija (posebno u sistemima s većim brojem slojeva)
3	Jednostavno održavanje jer svaki sloj ima dobro definisanu ulogu	Svi sistemi se ne preslikavaju direktno u slojevitu arhitekturu
4	Minimizirana je međusavisnost komponenata	
5	Podržava oblikovanje programske podrške na visokom apstraktnom nivou	
6	Podržava ponovno korišćenje	

Za razliku od klijent-server arhitekture, CORBA arhitektura podržava komunikaciju dva programa iako:

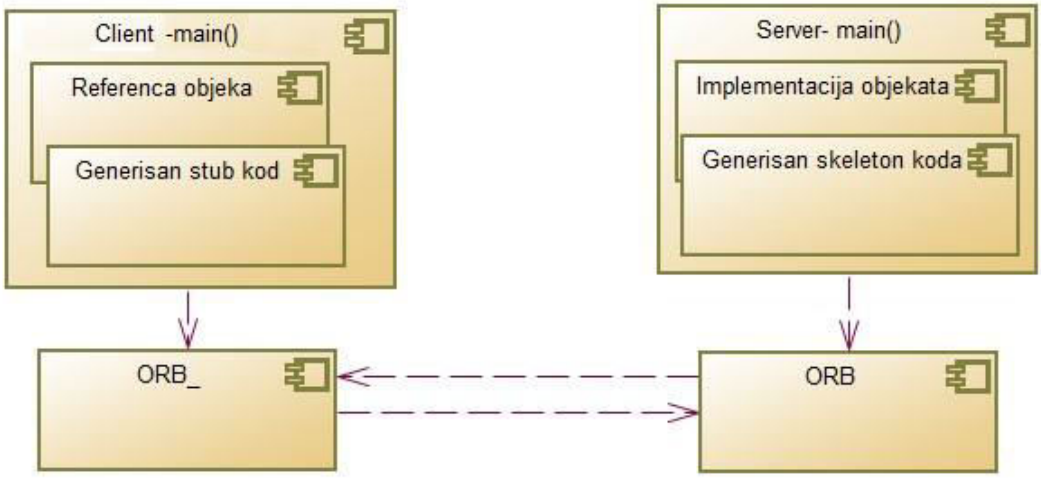
- Su piisana na različitim programskim jezicima (Java i C++ npr.)
- Se izvršavaju na različitim OS (Windows i Unix)
- Se koriste različiti mrežni protokoli (TCP/IP i Modbus)
- Se izvršavaju na različitom hardveru (Intel 32bit i Alpha CPU)
- Su korišćene različite mrežne tehnologije (Ethernet i Token ring mreže)

4.2.2 Način primene CORBA arhitekture na Home Security sistemu



sl.Dijagram CORBA arhitekture

Za implementaciju interfejsa, CORBA IDL se prevodi (kompajlira) na programski jezik klijenta i servera. Na strani klijenta ovaj kod se naziva „stub“, a na strani servera „skeleton“. Stub metoda inicira mehanizme povezivanja na ORB-u kojima se zahtev prosleđuje prema distribuiranom objektu. Klijent aplikacija, da bi zatražila servis od servera, poziva metode „stub“. Zahteve zatim obrađuje ORB i prosleđuje ih serveru preko „skeletona“. Skeleton metoda prevodi upit i parametre pristigle od strane klijenta te poziva izvršavanje metode na distribuiranom objektu. Objekat servera zatim prihvata zahtev i klijentu vraća odgovor.



sl. Implementacija CORBA-e

Da bi se gore skicirano u praksi primenilo, potrebno je uraditi sledeće:

1. **Pravljenje IDL fajla da bi se definisao interfejs aplikacije** – IDL obezbeđuju odvojene interfejse za operatvnih sisteme i programske jezike i dobijaju pristup svim servisima i komponenntama koje su zajedno povezani na ORB. IDL specificira objašnjenje za sve servise aplikacije.

2. **Prevođenje IDL fajla** – IDL prevodilac je obično sastavljen iz dva dela koja prestavljaju saradnju između klijenta i implementaciju servera, stub koda i skeleton koda.

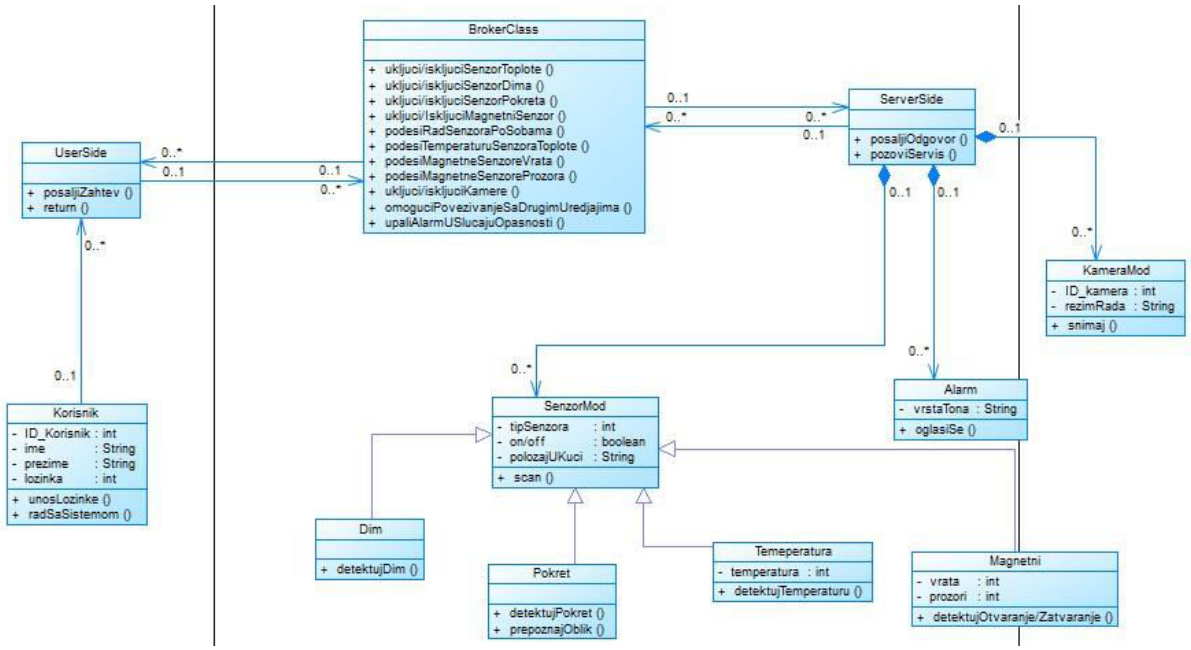
3. **Kompajliranje interfejsa**-Kad je IDL preveden na odgovarajući jezik svi interfejsi se sastavljaju i pripremaju za realizaciju objekta.

4. **Izrada/završetak implementacije** – Ako su klase nepotpune to mora da se napravi da se završi kompletna implementacija koda. Ako imamo promene hardvera ili bilo kakvog drugog problema, to sve mora se implementira.

5. **Kompajliranje implementacije –** U ovom procesu se uključuje kompajliranje samog koda koji je implementiran. Kljent je već omogućen da preko interfejsa upravlja resursima koji su za njega omogućeni i samim tim kod je prikladno implementiran.

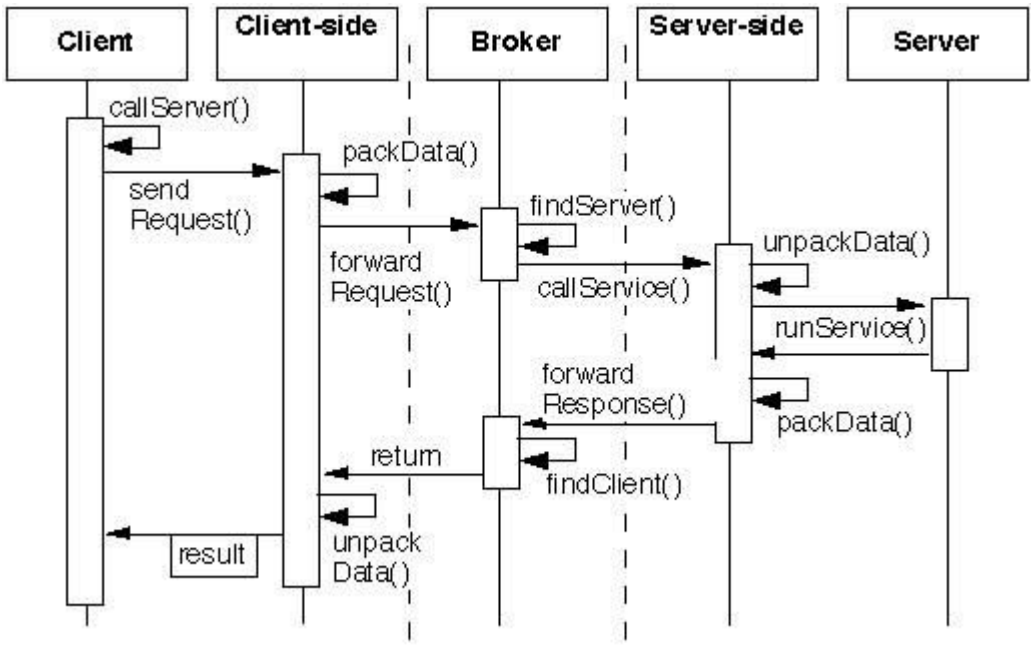
6. **Povezivanje Apikacije –** Ovde se vrši spajanje interfejsa sa kodom. Naravno glavna stvar je da se ovde preko ovoga uspostavlja konekcija između klijenta i servera. Tako da se sve sastavlja na ORB i samim tim aplikacija je kompletna.

7. **Pokretanje klijenta i servera –** Na kraju pokrećemo server na kojem će klijenti da funkcionišu sa serverom. Nakon toga se poreću „klijenti“ , i tako se komunicira sa serverom što smatramo gotovim proizvodom.



sl. CORBA dijagram klasa

Kao što možemo videti sa dijagrama, što je i prethodno dato u dijagramu arihtekture, postoje klijentska i serverska strana, kao i treći sloj koji je zadužen za komunikaciju između njih (u ovom slučaju klasa Broker). UserSide i ServerSide su klase koje omogućuju konkretnu "konverzaciju" i moraju da postoje na obe strane.

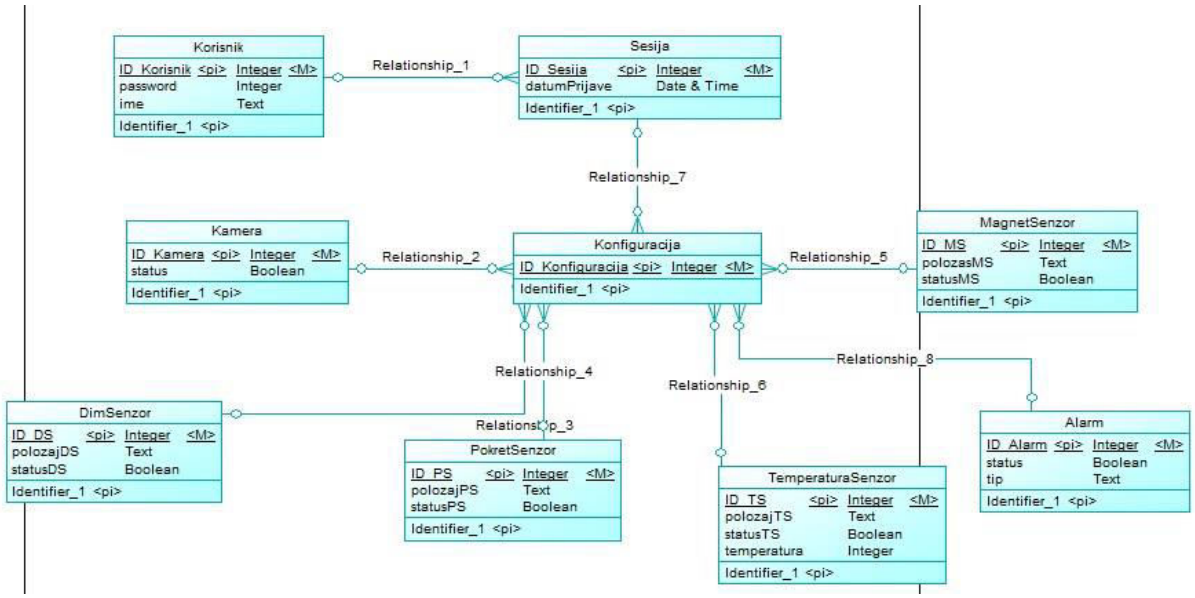


sl. Komunikacija između komponenti u CORBA arhitekturi

4.3 Struktura baze podataka

Jedan od glavnih delova svakog sistema svakako je baza podataka, tj direktorijum u kome se čuvaju svi podaci vezani za sam sistem. Home Security System poseduje sopstvenu bazu podataka, kao što je već prikazanu prethodno na dijagramima arhitekture.

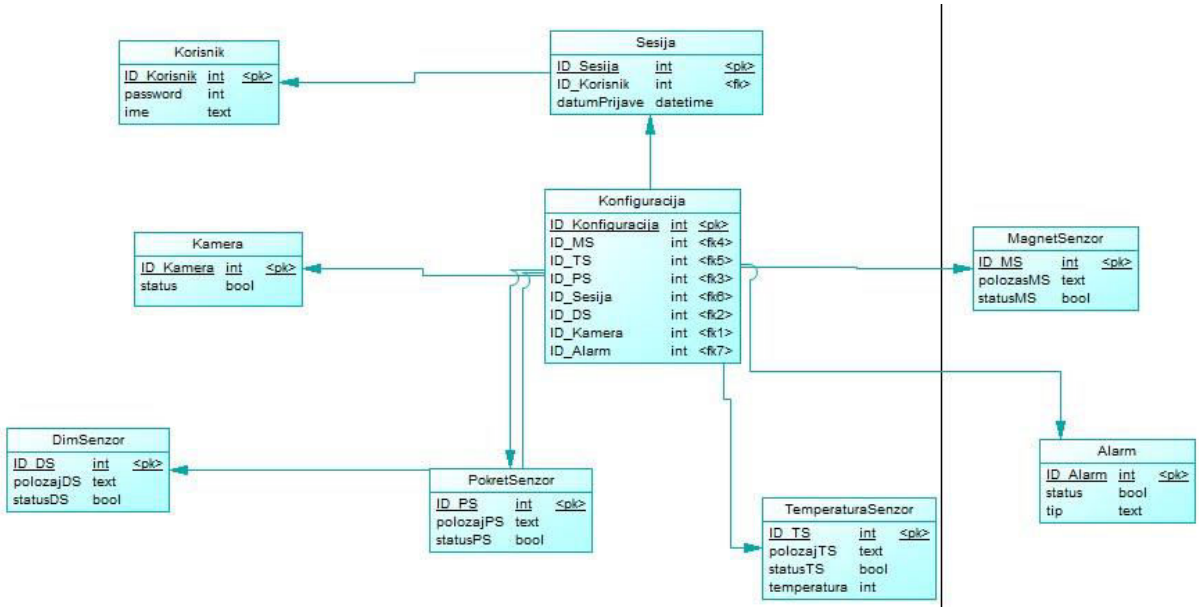
4.3.1 Konceptualni model baze podataka



sl. Konceptualni model

Konceptualni model nam daje strukturu baze podataka našeg sistema. Svaki entitet predstavlja jednu komponentu našeg sistema (senzore za dim, senzore za pokret itd.) , kao i atribute koje ta komponenta sadrži. Predstavljen je i entitet “Korisnik” koji čuva informacije o klijentu koji se prijavljuje na sam sistem i čijom prijavom otpočinje sesija kojoj se pamti datum i vreme. Konfiguracija svakog od delova našeg sistema, odnosno gore navedenih komponenti se čuva u entitetu “Konfiguracija”.

4.3.2 Fizički model baze podataka



sl.Fizički model

Fizički model predstavlja detaljniji prikaz baze podataka, a dobija se direktnim generisanjem konceptualnog modela. U fizičom modelu možemo videti učinak prethodno definisanih relacija u konceptualnom modelu.

5. Zahmanova matrica

Zahmanov okvir (framework) izumeo je sam John Zachman 1980. godine za IBM. Ovaj okvir pozajmljuje principe u arhitekturi i manufakturi iz biznis dizajna i obezbeđuje način posmatranja projekta iz različitih persepktiva prikazujući pritom i koje su veze između komponenata tog projekta. U današnjim složenim poslovnim okruženjima, mnoge velike organizacije teško reaguju na promene.

Ove poteškoće su posledica nedostatka internog razumevanja složenosti strukture i komponenata u različitim oblastima organizacije, gde su informacije o poslovanju zatvorene u glavama pojedinih radnika ili poslovne jedinice, a da nisu određene. Zachmanov okvir obezbeđuje sredstva za klasifikaciju arhitekture jedne organizacije. To je proaktivno poslovni alat, koji se može koristiti za modeliranje postojećih funkcija organizacija, elemenata i procesa - i pomoći u upravljanju poslovnih promena.

Okvir se oslanja na Zachmanovo iskustvo o tome kako se upravlja promenama u kompleksnim proizvodima, poput aviona i zgrada. Iako okvir svoju primenu može naći u arhitekturi informacionih sistema i široko je prihvaćen među analitičarima sistema, kao i dizajnerima baza podataka, John Zachman je naglasio da je primenjiv u svim sferama preduzetništva i nije ograničen striktno na informacionu arhitekturu. Okvir obezbeđuje konzistentan i sistematičan način opisivanja preduzeća I korišćen je u mnogim velikim organizacijama, kao što su Volkswagen, General Motors, Bank of America i Health Canada. Najlakši način da se razume Zachman Enterprise okvir arhitekture je videte ga kao klasifikacionu šemu vizuelno predstavljenu kao matricu, sa kolonama i redovima.

	Why	How	What	Who	Where	When
Contextual	Goal List	Process List	Material List	Organisational Unit & Role List	Geographical Locations List	Event List
Conceptual	Goal Relationship	Process Model	Entity Relationship Model	Organisational Unit & Role Relationship Model	Locations Model	Event Model
Logical	Rules Diagram	Process Diagram	Data Model Diagram	Role Relationship Diagram	Locations Diagram	Event Diagram
Physical	Rules Specification	Process Function Specification	Data Entity Specification	Role Specification	Location Specification	Event Specification
Detailed	Rules Details	Process Details	Data Details	Role Details	Location Details	Event Details

sl.Zahmanova matrica

Svaka ćelija u matrici predstavlja jedinstven model ili reprezentaciju projekta. Informacije u svakom redu matrice bi bile relevantne za konkretno lice koje taj projekat sagledava iz svog ugla. Okvir nudi niz opisnih reprezentacija ili modela koji su relevantni za projekat koji se opisuje. Svaka ćelija u tabeli mora biti u skladu sa ćelijama iznad i ispod nje. Sve ćelije u svakom redu takođe moraju biti usklađene međusobno, a svaka ćelija je jedinstvena. Kombinovane ćelije u jednom redu formiraju kompletan opis projekta iz tog pogleda.

Zachmanov okvir može da se posmatra kao sredstvo za stvaranje znanja i razjašnjenog razmišljanja, ali i kao pomoć u analizi i odlučivanju. To je strategijski-informaciono sredstvo koje može da pomogne oblikovanju organizacije.

Prednosti uključuju:

• Lako dostupne dokumentacije za projekte

• Sposobnost ujedinjavanja i integrisanja poslovnih procesa u okviru projekta

• Povećanje poslovne agilnosti, tako što će smanjiti složenost

• Redukovanje vremena za donošenje rešenja i smanjenje troškova isporuke uz maksimalno povećanje ponovne upotrebe

• Sposobnost za kreiranje i održavanje zajedničke vizije budućnosti između poslovne i IT zajednice.

Matrica se deli na 5 pogleda:

- Pogled planera - predstavlja izvršni rezime projekta i ovde spadaju aspekti kao što su ukupna veličina, cena, odnos prema životnoj sredini, itd.
- Pogled vlasnika – predstavlja plan arhitekte koji daje pogled sistema visokog nivoa iz perspektive vlasnika.
- Pogled dizajnera – predstavlja obično detaljniji tehnički pogled na nešto kao što su strukture podtaka, ponašanje, itd.
- Pogled inženjera – ovaj pogled je sličan kao i dizajnerski samo što se ističu implementacioni problemi i ograničenja kao što su alati, tehnologije, materijal, itd. Pogled programera –obuhvata bilo koju konfiguracionu, građevinsku ili razvojnu informaciju koja je potrebna.

Za svaki od navedenih pogleda važno je generisati svih šest perspektiva, među kojima su:

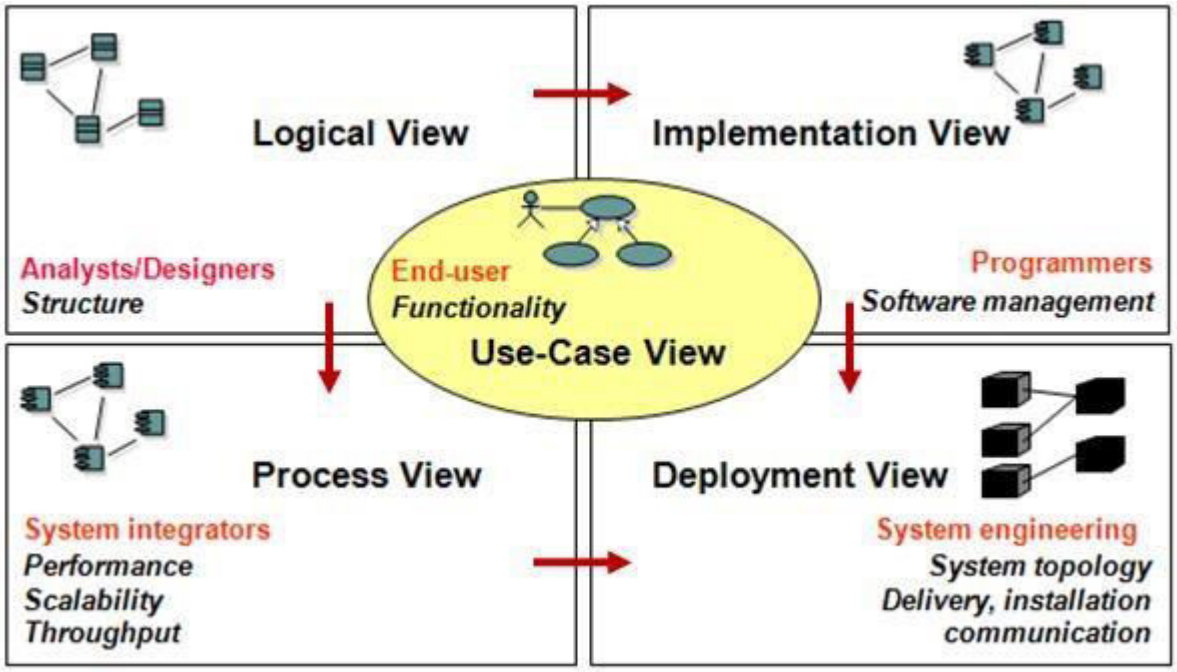
- ŠTA (opis podataka) – opisuje glavne elemente i veze među njima.
- KAKO (funkcioniše) – opisuje kako svaki element funkcioniše, i kako međusobno funkcionišu.
- GDE (opis mreže) – prikazuje razvojne aspekte elemenata sistema i svih zavisnosti među njima.
- KO (je uključen) – opisuje ljude koji su ukjučeni u pogled.
- KAD (se stvari obavljaju) – opisuje bilo koji vremenski aspekt sistema.
- ZAŠTO (se stvari dešavaju) - opisuje osnovne relacione zahteve pogleda.

	ŠTA	KAKO	GDE	KO	KAD	ZAŠTO
	PODACI	FUNKCIJE	MREŽA	LJUDI	VREME	MOTIVACIJA
OBIM	Home Security System zasniva se na podacima koji su opšte prihvaćeni za ovakve sisteme, i zasnivaju se na kompletnoj zaštiti doma. Vidi tačku 2.0	Potrebno je znati koji tim radi na razvoju, koja su ograničenja samog sistema, komunikaciju sa drugim sistemima, metodologiju razvoja itd. Vidi tačku 2.1 i 2.2	Sistem je integrisan u sam dom klijenta i komunicira sa senzorima putem prijemnika i predajnika. Potreban je Wi-Fi signal ukoliko mu pristupamo preko povezanih uređaja. Vidi tačku 4.1.1	Na projektu radi Predrag Jevtić kao glavni izvršilac na projektu. Konsultacije sa asistentom N.Gavrilovićem.	Projekat je potrebno završiti i predati najkasnije do 5. Aprila , 2021 god.	Videti tačku 2.0
BIZNIS	Sa aspekta biznis logike, podaci se čuvaju u bazi podataka, koja je detaljno objašnjena u tački 4.3	Home Security System se bazira na pružanju totalne zaštite svakog doma, odnosno vlasnicima ovog sistema.	Sa aspekta biznis logike, odnosno iz ugla vlasnika, sistem se nalazi u kući. Vidi tačku 1.0	Dodeljiivanje zadataka u okviru procesa razvoja softvera izučava se na predmetu SE325.	Kreiranje vremenskog rasporeda projekta potpada pod predmet SE325.	Kreiranje plana pravila i poltike potpada pod predmet SE325.
SISTEM	Kako se kreću podaci između komponenti možete videti u okviru tačke 5.0	Da bi bolje razumeli samo funkcionisanje sistema, potrebno je upoznati se sa njegovom arhitekturom. Vidi tačku 4.1 i 4.2	Potrebno je razumeti način interakcije i saradnje komponenata odrađenog procesa. Vidi tačku 5.3	Korisnik sistema uvek treba biti u centru pažnje prilikom kreiranja studije slučaja korišćenja. Vidi tačku 5.1	Kreiranje vremenskog rasporeda procesa potpada pod predmet SE325.	Kreiranje biznis plana potpada pod predmet SE325.
TEHNOLOGIJE	Sa tehnološke tačke gledišta u pogledu podatka bitan je fizički model baze podataka, njega možete videti u tački 5.2	Pogled na sistem arhitekturu tehnologije gde vidimo uređaje kojim pristupa korisnik, raspored servisa, veze između komponenti… Detaljnije pod tačkom 4.1.1 i 4.2.1	Kako su te komponente raspoređene u okviru servera i kako one komuniciraju pokazaće nam pogled raspoređivanja. Vidi tačku 5.4	Prototip ovog sistema nije deo ovog predmeta, već samo arhitektura i funkcionisanje sistema.	Proveru ispravnosti sistema možete videti u tački 6.0	Kreiranje biznis plana potpada pod predmet SE325.

PROGRAMER	U sistemu koristimo relacionu bazu podataka koja je deo samog sistema zaštite doma.	S tačke gledišta kodera treba uzeti u obzir kako će se implementirati komponente sistema. Tačka 5.5	Kod mrežne komunikacije veoma je bitno koji se protokoli koriste.	Sigurnost podataka sistema je veoma birna jer ona predstavlja ključ optimizacije procesa.	Detaljna definicija vremena potpada pod predmet SE325.	Strategija razvoja softvera je W model. Vidi tačku 2.2

6. "4+1" pogled

U slučaju razvoja kompleksnih sistema, nije moguće jedinstvenim modelom predstaviti ključne funkcionalnalne i nefunkcionalne zahteve sistema, tako da on bude razumljiv svim stejkholderima. Široko prihvaćeno stanovište je da razvoj arhitekture kompleksnih sistema treba da bude zasnovan na brojnim odvojenim, ali međusobno povezanim pogledima, pri čemu svaki od njih opisuje poseban aspekt arhitekture. Ideja potiče još od Parnasa 1970. godine, ali je u širokoj upotrebi tek pošto je Kruchten objavio svoj model pogleda na razvoj softverske arhitekture, poznat pod nazivom "Architectural Blueprints 4+1". Model je postao i sastavni deo RUP radnog okvira. Iz razloga jer ovim pogledom mogu da se obuhvate sve bitne karakteristike našeg sistema, on će biti korišćen u daljoj realizaciji.



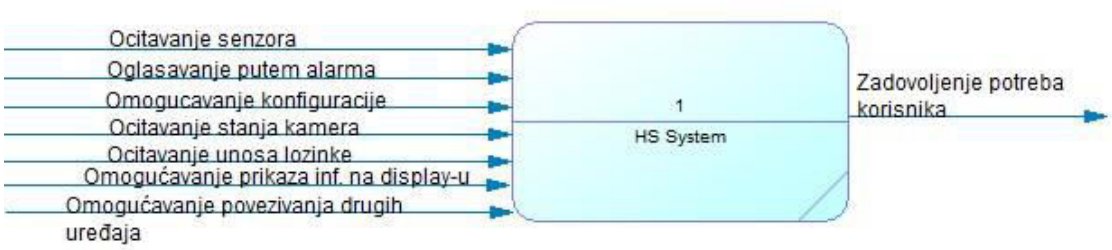
- Logički pogled opisuje funkcionalne zahteve sistema. Model dizajna je njegov sastavni deo i sadrži glavne pakete dizajna, podsisteme i klase.
- Procesni pogled, oslikava aspekte dizajna koji se odnose na konkurentnost i sinhronizaciju, u terminima procesa i niti.
- Pogled raspoređivanja, opisuje mapiranje softvera i hardvera i odražava aspekt distribuiranosti.
- Implementacioni pogled, opisuje organizaciju statičkih modula softvera u radnom okruženju, u terminima paketa i slojeva.

Pogled slučaja upotrebe, opisuje ključne scenarije ili slučajeve upotrebe, koji su putokaz za dizajn arhitekture, ali i sredstvo validacije drugih pogleda.

6.1 Pogled slučaja upotrebe

Detaljan pogled slučaja upotrebe dat je na početku samog dokumenta tako da neće opet ovde biti predstavljan. U nastavku će biti dat prikaz zadataka koje sistem treba da obradi, da bi slučajevi korišćena ispravno bili zadovoljeni.

6.1.1 Zadaci sistema



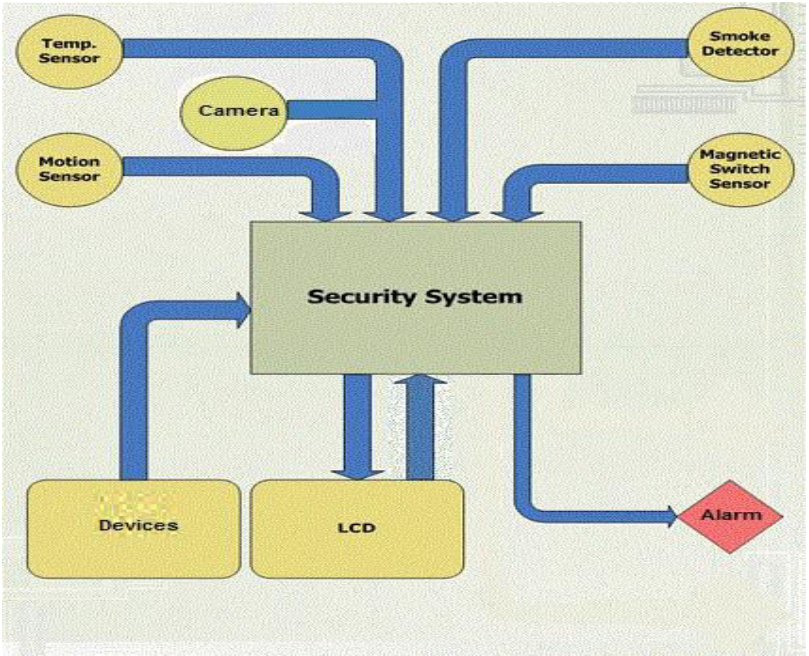
sl.Prikaz funkcija sistema

Zadatak	Objašnjenje
1. Očitaj senzore	Sistem za zaštitu ima zadatak da očita senzore koji su raspoređeni po domu i koji detektuju promene u temperaturi, detektuju dim, pokrete, kao i magnetne senzore postavljene na vratima i prozorima.
2. Oglasi se putem alarma	Da bi se sitem oglašavao potrebno je pre svega da očitava senzore, a zatim i da bude konfigurisan na koju jačinu senzora će reagovati, odnosno uključivati alarm.

3. Omogući konfiguraciju	Konfiguraciji pristupa korisnik sistema, koji podešava sam sistem, pali ga ili gasi, uključuje ili isključuje mod-ove,kamere itd.-na displey-u ili drugim uređajima.
4. Primaj signal kamere	Sistem mora da omogući korisnicima prikaz onoga što snimaju kamere koju su postavljenje van kuće – na displey-u ili drugim uređajima.
5. Očitaj unos lozinke	Sistem treba da očitava unos lozinke korisnika i zavisno od ispravnosti unetih podataka odlučuje o daljem radu, odnosno, da li će tražiti ponovni unos ili će proslediti korisnika do funkcija sistema.
6. Prikazivanje informacija	Sistem treba sve informacije da prikazuje na display-u prek koga korisnik vrši određene radnje
7. Povezivanje	Sistem treba da bude u mogućnosti da se poveže sa spoljašnjim uređajima i tako korisniku omogući lakše korišćenje

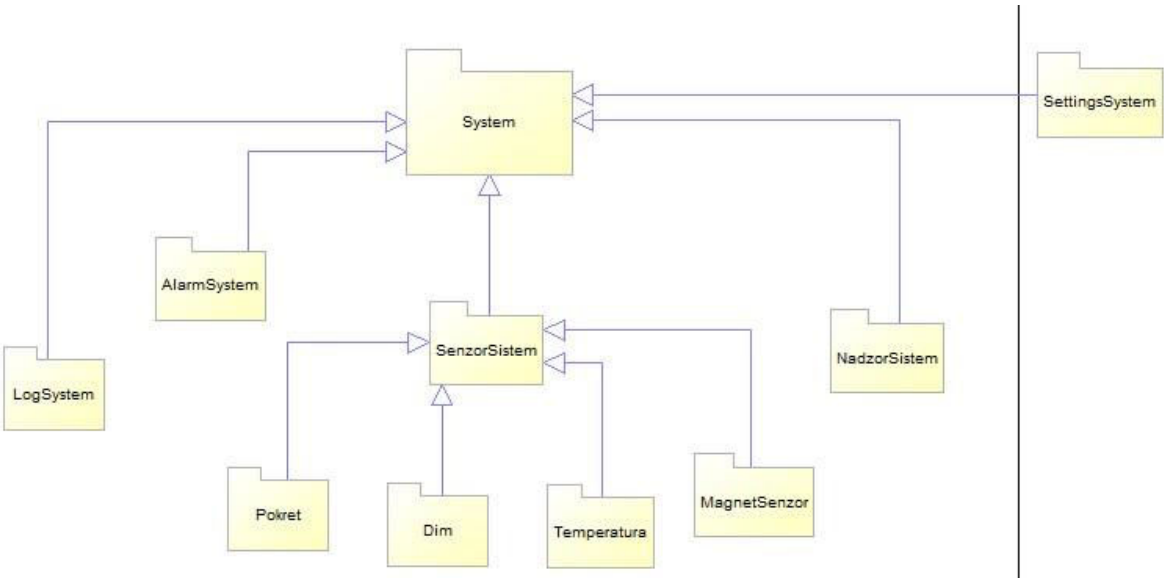
6.2 Logički model pogleda

Model dizajna nam daje najopštiji prikaz sistema koga razlažemo na podsisteme, a onda i na klase.



Sa modela dizajna možemo da vidimo da su ulazni parametri u sistem razni senzori,kamera, kao i unosi sa uređaja. Sam sistem obezbeđuje LCD display preko kojeg je takođe moguća interakcija sa sistemom, a kao glavni izlaz iz sistema naznačen je alarm.

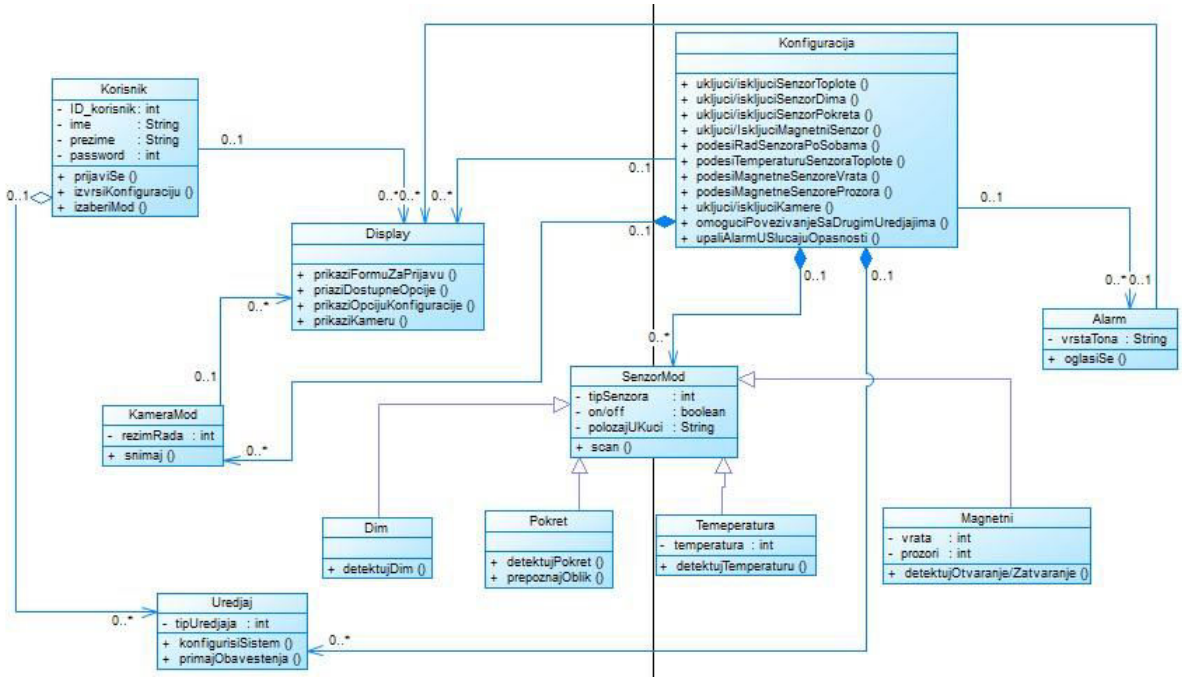
6.2.1 Model podsistema



sl.Model podsistema

Na gornjem dijagramu je dat prikaz podsistema našeg celog sistema. Vidimo da su to sistemi za logovanje, koji omogućava zaštićen pristup sistemu. Zatim sistem koji aktivira alarm. Nakon toga sistem za senzore, koji se dalje opet deli na sistem za dim, sistem za pokret, sistem za temperaturu i sistem za magnete senzore koji se postavljaju na vrata i prozore. Imamo sistem za nadzor, koji se odnosi na kamere. I postoji sistem za konfiguraciju koji nam omogućava upravljanje celokupnim sistemom, kao i svim podsistemima.

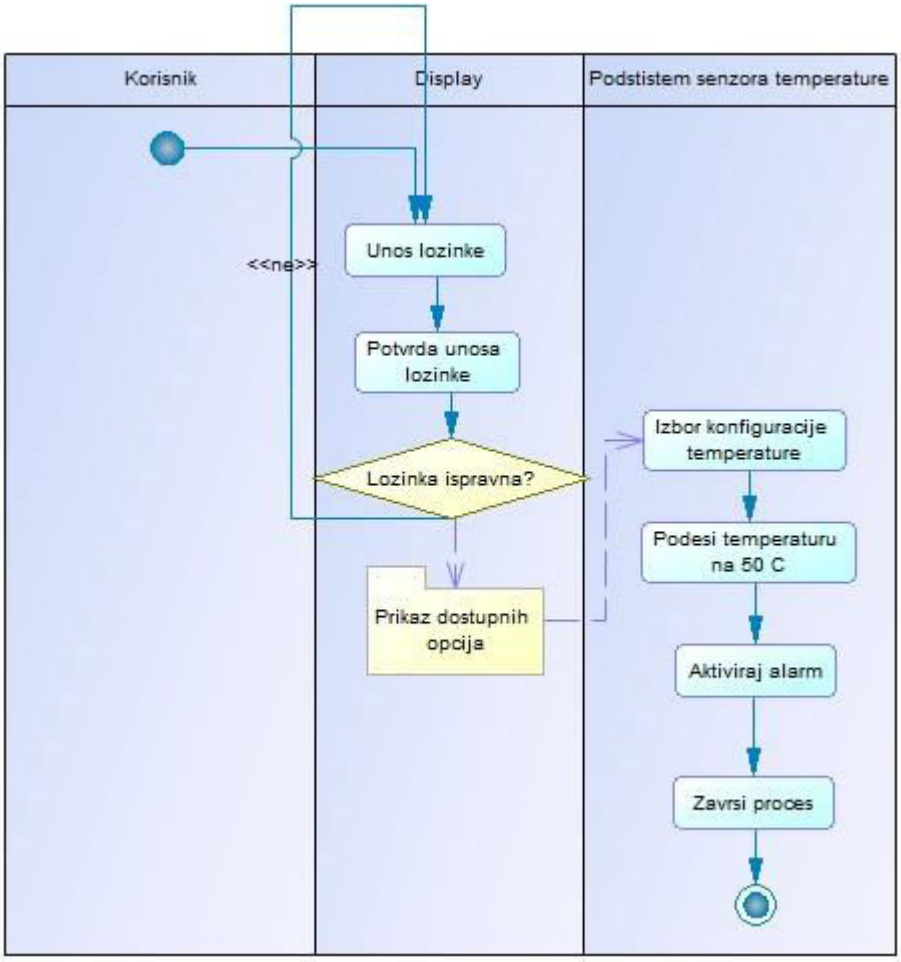
6.2.2 Dijagram klasa



sl.Dijagram klasa

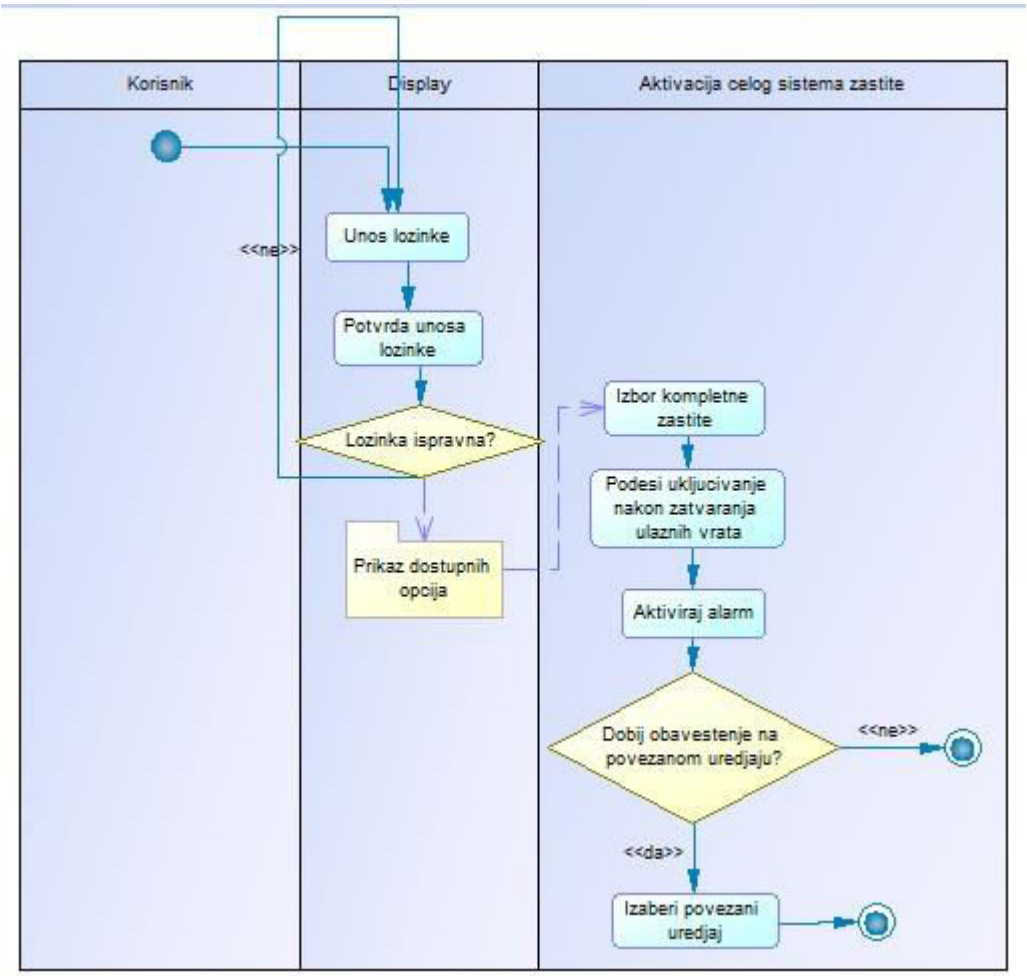
Prethodno dat dijagram klasa daje nam kompletan pregled načina funkcija celog sistema i njihove povezanosti. Glavna klasa svakako je konfiguracija, preko koje se podešavaju i senzori i kamera i alarm, a sve se to vrši preko klase koja omogućava prikaz informacija na display-u, ili ipak sa druge strane na nekom uređaju.

6.3 Procesni pogled



sl.Primer procesa aktiviranja moda za senzore temperature

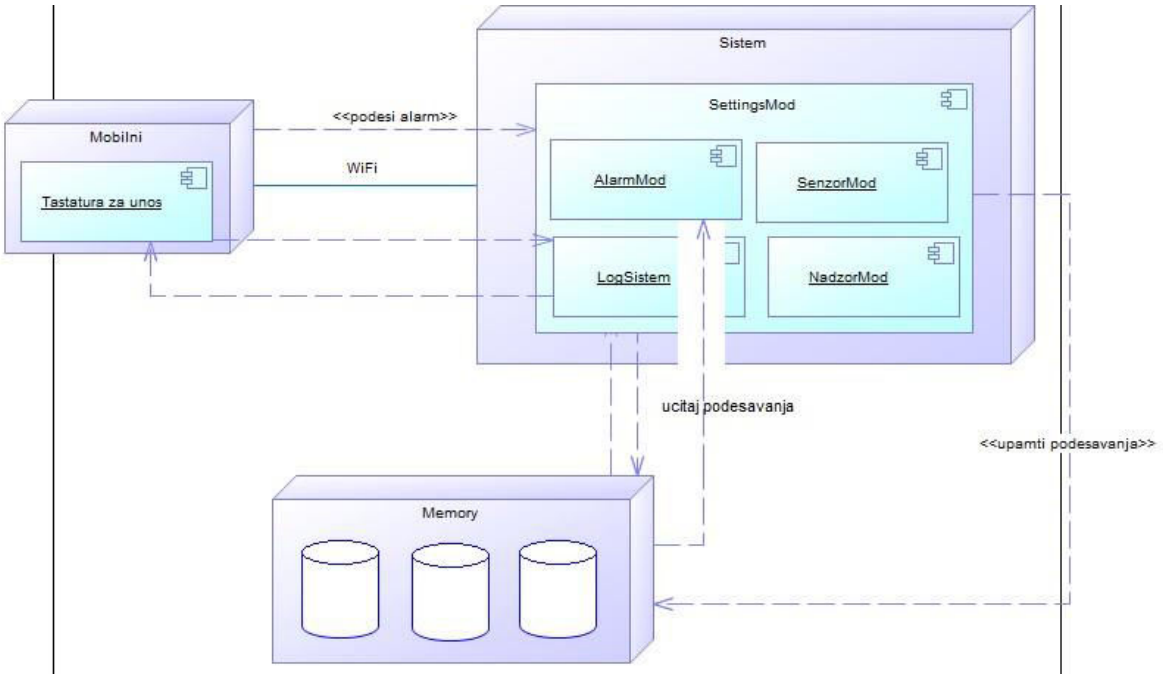
U ovom slučaju prikazan je model i način aktiviranja senzora za temperaturu. Vidimo da je pre samog pristupa samo sistemu potrebno da se utvrdi da li je lozinka koju je korisnik uneo (bilo da je to u pitanju direktno na displey-u ili preko mobilnog npr.) ispravna, i da se tek nakon toga dobija spisak dostupnih opcija. Korisnik na primeru bira opciju za aktiviranje određenog mod-a sistema, podešava da se alarm aktivira kada temperatura dostigne 50 stepeni i završava svoj proces.



sl.Primer procesa aktivranja kompletne zaštite doma

Kada napuštamo svoj dom, svakako sam potrebni svi vidovi zaštite koje sistem pruža. Zato na glavnom uređaju prilikom izlaska unosimo svoju lozinku, dobijamo spisak dostupnih opcija, i biramo da se sistem aktivira nakon našeg izlaska, odnosno nakon zatvaranja ulaznih vrata. Takođe, biramo opciju da li želimo obaveštenje na svom mobilnom ukoliko dođe do aktivacije sistema dok smo odsutni.

6.4 Pogled raspoređivanja



sl.Deployment model sistema

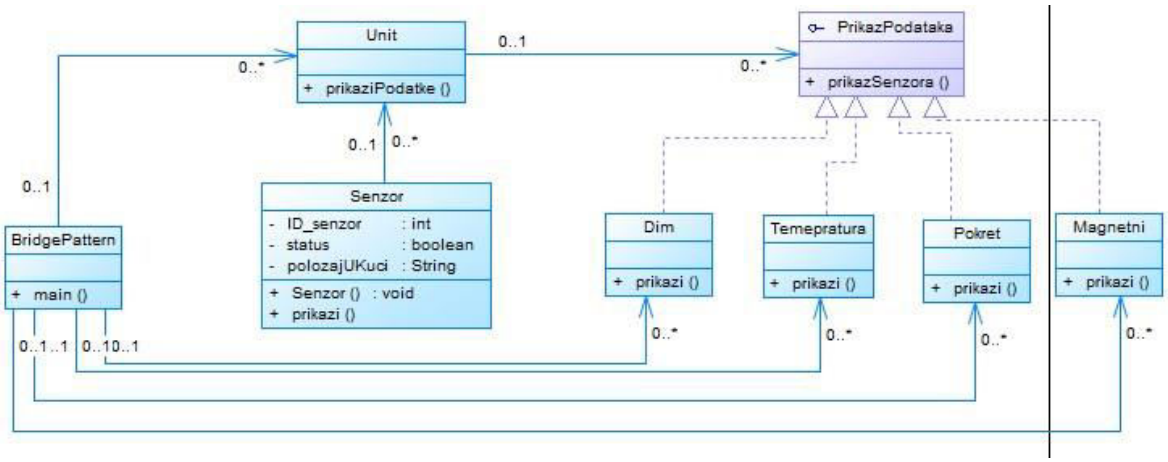
Sistem poseduje sopstvenu memoriju u kojoj čuva podatke o korisnikovoj lozinci i njegova podešavanja vezana za sistem. Na dijagramu je prikazan način na koji korisnik može putem željenog uređaja da se prijavi na sistem i izvrši konfiguraciju. Mobilni je WiFi signalom povezan sa samim sistemom, i nakon unosa lozinke aktivira se podsistem za logovanje,koji u memoriji proverava podatke i zavisno od njihove ispravnosti vraća korisniku poruku. SettingsMod je odgovoran za rad svih ostalih modova, i preko njega se izvršavaju podešavanja za ceo sistem. Takođe, i ta podešavanja se pamte u memoriji, a podsistem za alarm ih učitava, i zavisno od toga se i aktivira kada je potrebno.

6.5 Implementacioni pogled

Implementacioni pogled nam daje uvid u paterne koji će se koristiti za samu implementaciju HomeSecurity sistema. Biće prikazane prednosti i mane uzoraka, kao i način njihove implementacije na dijagramima klasa.

6.5.1 Bridge Pattern

Bridge pattern se koristi kada želimo da razdvojimo nivo apstrakcije od njegove implementacije, tako da ove dve celine mogu samostalno da rade. Ovaj uzorak predstavlja tip strukturnih uzoraka, i vrši ovakav tip razdvajanja tako što predstavlja "most" između dve forme.



sl.Primena Bridge pattern-a

Prednost ovog tipa uzorka je svakako razlaganje programa na manje jedinice, i njihov nezavisan rad koji se povezuje jednom klasom, odnosno Bridge patternom. Nedostatak je ukoliko postoji veliki broj jedinica koje treba da naslede sam interfejs. U nastavku će biti prikazana konkretna primena.

Najpre je potrebno definisati interfejs, i metodu koja nam treba. Uzećemo npr. metodu koja radi paljenje i gašenje senzora:

```
public interface Ukljucivanje {
    public void status(boolean status);

}
```

Zatim, potrebno je za svaki senzor posebno napraviti klasu, i overrajdovati ovu metodu:

```
public class Dim implements Ukljucivanje{

    @Override
    public void status(boolean status) {
        if (status == true) {
            System.out.println("Ukljucili ste senzor dima");
        }
        else {
            System.out.println("Iskljucili ste senzor dima");
        }
    }
}
public class Magnetni implements Ukljucivanje {

    @Override
    public void status(boolean status) {
        if (true) {
            System.out.println("Ukljucili ste magnetne senzore");
        }
        else {
            System.out.println("Iskljucili ste magnetne senzore");
        }
    }

}
```

```
public class Pokret implements Ukljucivanje{

    @Override
    public void status(boolean status) {
        if (status == true) {
            System.out.println("Ukljucili ste senzor pokreta");
        }
        else {
            System.out.println("Iskljucili ste senzor pokreta");
        }
    }

}
public class Temperatura implements Ukljucivanje{

    @Override
    public void status(boolean status) {
        if (status == true) {
            System.out.println("Ukljucili ste senzor temeprature");
        }
        else {
            System.out.println("Iskljucili ste senzor temperature");
        }
    }

}
```

Zatim kreiramo klasu Unit koja će nam dati konkretnu metodu koju pozivamo na kraju, prilikom čitanja statusa senzora:

```
public abstract class Unit {

    protected Ukljucivanje uk;

    protected Unit(Ukljucivanje uk) {
        this.uk = uk;
    }

    public abstract void status();

}
```

A onda i klasu Senzor koja definiše jednostavan konstruktor koji se na kraju poziva, kao i prethodnu metodu status() :

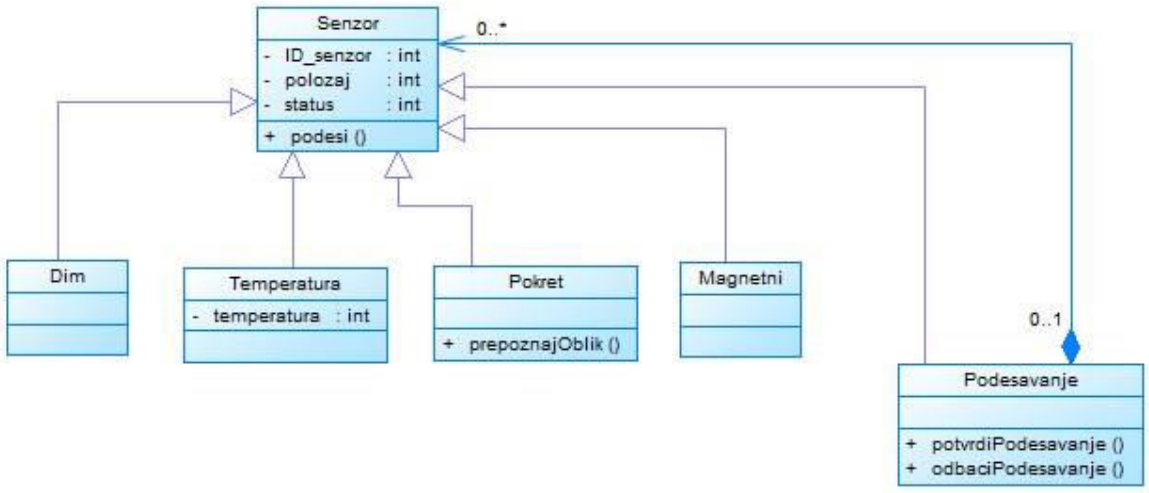
```
public class Senzor extends Unit{

    private boolean status;

    public Senzor(boolean status, Ukljucivanje uk) {
        super(uk);
        this.status = status;
    }

    @Override
    public void status() {
        uk.status(status);
    }
```

U klasi Main dat je poziv svega prethodno definisanog:

```
public class Main {

    public Main() {

        Unit senzorDim = new Senzor(true, new Dim());
        Unit senzorTemperatura = new Senzor(false, new Temperatura());

        senzorDim.status();
        senzorTemperatura.status();


    }
    public static void main(String[] args) {
        new Main();
    }
}
```

6.5.2 Composit Pattern

Composit pattern se koristi kada grupu određenih objekata , tretiramo kao jedan objekat.



sl.Primena Composit pattern-a

Primena ovog patterna na ovom sistemu zamišljena je tako da korisnik može podešavati više vrsta senzora, i svaka ta vrsta predstavlja jedan objekat. Nakon podešavanja određenog broja vrsta senzora, zajedno za sve njih kažemo da potvrđujemo ili odbacujemo podešavanja, što je konkretna primena pattern-a.

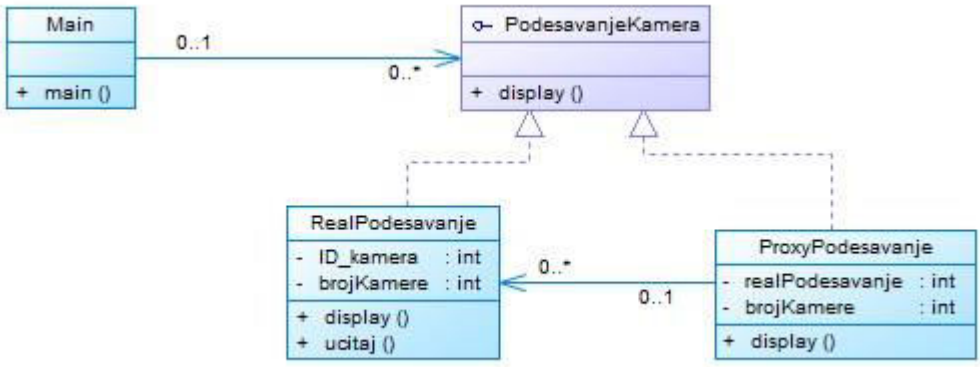
Kao što sam već naveo glavna prednost korišćena ovog patterna je ta što dozvoljava tretitranje više objekata kao jedan objekat. Takođe, ovaj šablon predstavlja i hijerarhiju objekata.

Nedostatak je to što je u nekim slučajevima dosta teško koristiti composite za samo određene slučajeve iz hijerarhije , i u tom slučaju moraju da se koriste real-time provere nakon pokretanja programa.

Testiranje i primena ovog paterna data je u tački 7.10

6.5.3 Proxy Pattern

Ovaj šablon je ustvari klasa koju poziva klijent da bi pristupio nekom određenom objektu koji je komplikovano direktno instancirarati, ili skupo itd.



sl.Primena Proxy pattern-a

Prednosti su kao što sam naveo indirektan pristup nekom određenom velikom objektu, kao i sprečavanje njegovog dupliranja. Nedostatak je to što ne možemo manipulisati objektom koji je već definisan, već je Proxy samo put do tog objekta. U nastavku će biti prikazana konkretna primena.

Najpre je potrebno definisati interfejs koji sadrži metodu sa kojom radimo. Stavićemo da to npr bude prikazivanje kamera, tj njihovih snimaka:

```
public interface Kamera {

    public void prikaziKameru();

}
```

Zatim, potrebno je definisati klasu koja upravlja kamerama, i implementira prethodno definisani interfejs:

```
public class RealKamera implements Kamera{

    private int brojKamere;

    @Override
    public void prikaziKameru() {
        System.out.println("Prikazivanje kamere broj: " + brojKamere);
    }

    public RealKamera(int brojKamere) {
        this.brojKamere = brojKamere;
    }

}
```

Prethodna klasa radi direktno sa kamerama, i vrši njihov prikaz korisniku. Međutim, da bi izbegli direktan pristup tako bitnim klasama, koristimo sledeći način:

```
public class ProxyKamera implements Kamera{

    private RealKamera rk;
    private int brojK;

    public ProxyKamera(int brojK) {
        this.brojK = brojK;
    }

    @Override
    public void prikaziKameru() {
        if (rk == null) {
            rk = new RealKamera(brojK);
        }

        rk.prikaziKameru();
    }
}
```

Prethodna klasa implementira takođe početni interfejs, ali u sebi sadrži rad RealKamera klase. Preko ProxyKamera ćemo pristupati našim kamerama i na taj način izbeći direktan pristup real klasi. U narednoj, nalazi se poziv svega prethodno objašnjenog:

```
public class Main {

    public Main() {

        Kamera k1 = new ProxyKamera(1);
        Kamera k2 = new ProxyKamera(2);

        k1.prikaziKameru();
        k2.prikaziKameru();

    }

    public static void main(String[] args) {
        new Main();
    }
}
```

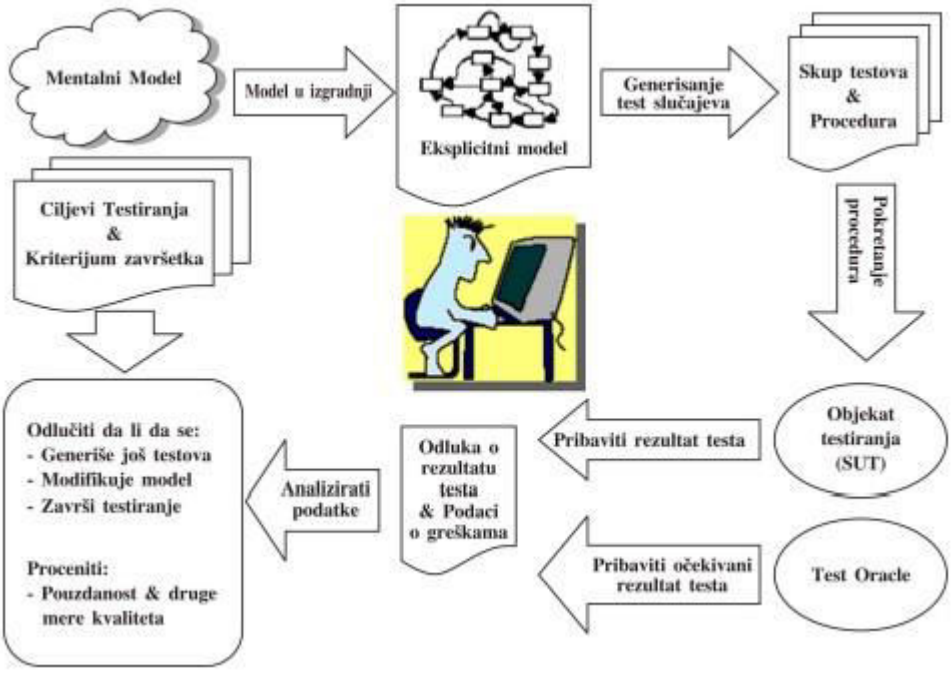
7. Testiranje

Prema IEEE 829-2008 standardu baziran je dokument master test plan. Prema ovom standardu master test plan bi trebao da sadrži sledeće elemente:

- Uvod
- Stavke testiranja
- Stavke koje će biti testirane Stavke koje neće biti testirana
- Pristup tesitranju
- Pass/Fail kriterijum
- Zadaci testiranja
- Testiranje isporuke Zahtevi okruženja
- Odgovornosti
- Osoblje zaduženo za testiranje
- Raspored testiranja
- Rizici
- Odobrenja

7.1 Uvod

Testiranje je aktivnost koja se izvodi identifikacijom defekata i problema radi procene kvaliteta proizvoda, kao i za njegovo unapređenje. Testiranje se više ne posmatra kao aktivnost koja počinje samo posle završetka faze kodiranja i ima ograničenu svrhu otkrivanja nedostataka. Danas se testiranje softvera posmatra kao aktivnost koja bi trebalo da obuhvati celi razvojni proces i važan je deo aktuelnog konstruisanja proizvoda.



sl.Glavne faze u testiranju softvera na bazi modela

Čak i posle uspešnog završetka obimnog testiranja, softver i dalje može sadržati greške. Lek za softverske greške nakon isporuke je obezbeđivanje korektivnih akcija održavanja softvera.

7.2 Stavke koje će biti testirane

R.B	Test slučaj
1	Prijava klijenta na sistem i unos lozinke
2	Izbor senzor mod-a
3	Rad sa senzorima za toplotu
4	Rad sa magnetnim senzorima
5	Rad sa senzorima za dim
6	Rad sa senzorima za pokret
7	Izbor kamera mod-a
8	Rad sa kamerom

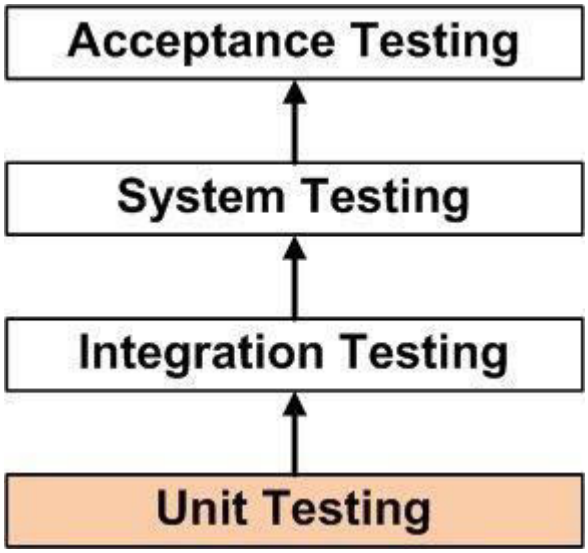
Nijedno testiranje ne može da obuhvati stoproscentno testiranje sistema, odnosno svih njegovih funkcija, tako da će ovde biti testirane samo neke od najbitnijih. Takođe, ova tabela će možda u nastavku i biti proširena nekim dodatnim funkcijama koje će biti testirane, sve u svrhu što kompletnijeg sistema.

7.3 Stavke koje neće biti testirane

U ovom delu dokumenta neće biti testirana funkcionalnost sistema kada mu se pristupa sa nekog drugog uređaja. U delu oko arhitekture je naveden deo i pružena mogućnost povezivanja samog sistema sa drugim uređajima, ali za to je svakako potrebno da postoji posebna aplikacija instalirana na tim uređajima koja će putem Wi-Fi signala komunicirati sa sistemom. Definisanje arhitekture i načina rada takve aplikacije bi bio sasvim nov projekat, tako da će se i samo njeno testiranje obaviti neki drugi put.

7.4 Pristup/strategija

Strategijom se daje prikaz plana izrade test dokumenta. Postoji više vrsta testiranja koje se mogu primeniti na sam naš sistem, a neke od njih će i biti korišćene u daljem radu:

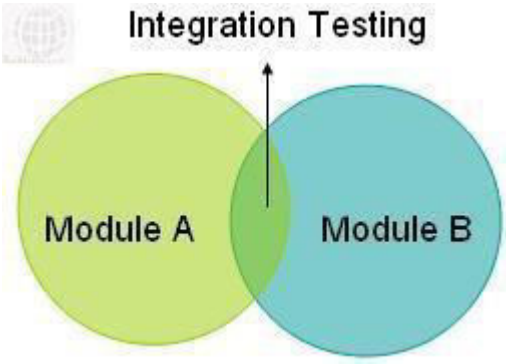


sl. Strategija testiranja sistema

- Jedinično testiranje verifikuje funkcionisanje u izolaciji softverskih delova koji se nezavisno mogu testirati. U zavisnosti od konteksta, to mogu biti individulani podprogrami ili veće komponente kreirane od tesno povezanih jedinica. Testiranje jedinice se precizno definiše standardom - IEEE Standard for Software Unit Testing (IEEE1008-87), koji takođe opisuje integralni pristup sistematičnom i dokumentovanom testiranju jedinice. Tipično, testiranje jedinice se dešava sa

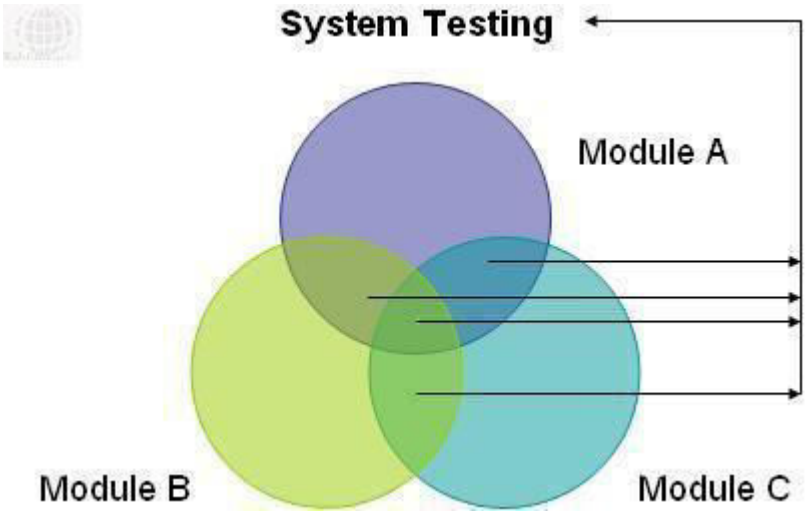
pristupom kodu koji se testira i sa podrškom alata za debagovanje, a može uključivati programere koji su pisali kod.

- Integraciono testiranje je proces verifikovanja interakcije između softverskih komponenti. Klasične strategije integralnog testiranja, kao što su top-down ili bottomup, se koriste kod tradicionalnog, hijerarhijski strukturiranog softvera. Moderene sistematične integrativne strategije su više arhitektonski struktuirane, što implicira integrisanje softverskih komponenti ili podsistema na bazi identifikovanih funkcionalnih delova. Integraciono testiranje je kontinualna aktivnost, u svakoj fazi u kojoj softver-inženjeri moraju umesto posmatranja na nižem nivou da se koncentrišu na perspektivu na integralnom nivou. Osim za male, jednostavnije softvere, sistematične, integralne test strategije se obično izvode putem testiranja svih komponenata zajedno u istom trenutku.



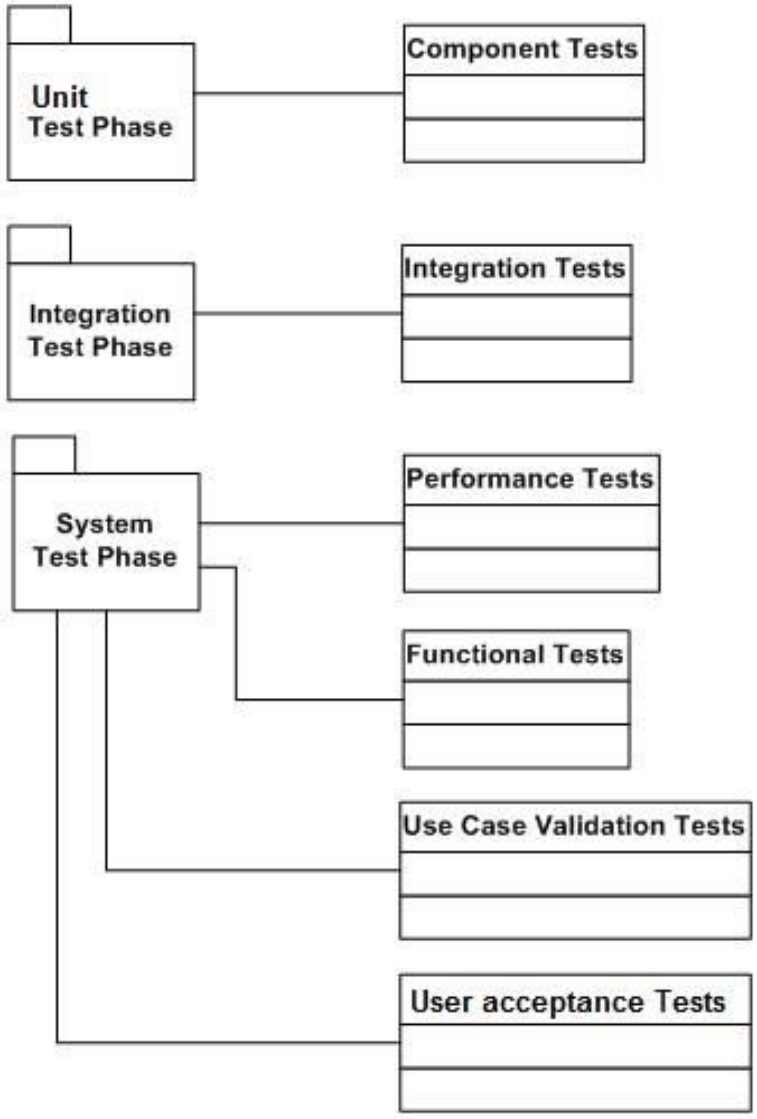
sl. Prikaz integracionog testiranja

- Sistemsko Testiranje se bavi ponašanjem celog sistema. Većina funkcionalnih otkaza trebalo bi da su već identifikovani tokom testiranja jedinice i integracionog testiranja. Sistemsko testiranje se obično razmatra prikladnim za poređenje sistema sa nefunkcionlanim sistemskim zahtevima, kao što su bezbednost, brzina, preciznost ili pouzdanost. Spoljašnji interfejs ka drugim aplikacijama, pomoćnim programima, hardverskim uređajima, ili operativnim okruženjima se takođe razmatra na ovom nivou.



sl. Prikaz sistemskog testiranja

- Test prihvatljivosti proverava ponašanje sistema prema zahtevima kupaca, bez obzira kako oni bili iskazani. Kupac podnosi ili specificira tipične zadatke radi provere da li su njihovi zahtevi ispunjeni. Ova aktivnost testiranja može, ali i ne mora, da uključuje razvojni tim sistema.



sl. Faze testiranja Home Security sistema

7.5 Potrebno osoblje i obuka

Pošto je svo osoblje zaduženo za razvoj softvera podeljeno u timove, tako će I ovde postojati tim koji će se isključivo baviti testiranjem. Zapravo, testiranje će nadgledati QA menadžer koji će u timu imati dvoje testera jer se više isplati imati dvoje testera koji će raditi puno radno vreme nego više njih koji će raditi skraćeno radno vreme. S obzirom na to da se zapošljavaju testeri koji su upoznati sa alatom za testiranje I koji imaju najmanje 5 godina iskustva sa ovim alatima, nije potrebna posebna obuka u tvoj sferi znanja. QA menadžer će prisustvovati isključivo sastancima, dok će testeri biti zaduženi za testiranje I pisanje dokumentacije vezane za testiranja. Pisanjem dokumentacije QA menadžer će moći da vodi efikasnu

evidenciju o aktivnostima zaposlenih kao I o vremenu potrebnom za izradu testova. Evidencija će sadržati I greške, uslove I rezultate testiranja. Za Junit testove biće zaduženi programeri još u periodu implementacije sistema kako bi se testeri fokusirali na integralno testiranje I sistemsko testiranje. Ova odluka je doneta iz razloga što je testerima obaveza da pišu detaljnu dokumentaciju za vreme testiranja. Na ovakav način, testeri će dobiti dokumentaciju vezanu za softver na koju samo dodaju potrebne elemente nakon ova dva testiranja.

ULOGA	IME I PREZIME	ODGOVORNOST
QA menadžer	Predrag Jevtić	Vođenje evidencije i nadgledanje čitavog procesa testiranja
		Pružanje stručnih saveta
		Izrada plana testiranja
Tester 1	Marko Damnjanović	Testiranje softvera I pisanje dokumentacije
Tester 2	Nemanja Simić	Testiranje softvera I pisanje dokumentacije

7.6 Zadaci testiranja

Home Security sistem je sistem koji njegovom korisniku treba da pruži totalnu sigurnost i kada je u domu i kada je odsutan. Zbog toga ovaj sistem mora da bude izrađen tako da ima što manje grešaka. Zadatak testiranja ovakvog sistema jeste da uoči kvarove i probleme koji mogu da utiču na rad nekih od mod-ova sistema, nemogućnosti očitavanja senzora, kvarova u kamerama, grešaka oko oglašavanja alarmo itd. , otkloni ih i obezbedi klijentima pouzdan sistem za zaštitu.

7.7 Kriterijumi za obustavljanje testiranja

U toku testiranja može da se desi da dođe do određenih situacija koje na neki način onemogućavaju nastavak testiranja. U tim slučajevima potrebno je obustaviti testiranja. Neke od grešaka koje treba predvideti su:

- Pad sistema
- Pojava greške visokog prioriteta

- Pojava kritične greške

Za predviđene kriterijume potrebno je dati rešenja koja će se u tim situacijama iskoriste. Naime, ukoliko dođe do pada sistema, testeri će obustaviti testiranje I pokušati da povrate sistem u stanje u kom je bilo pre pada sistema. Kako bi ovo bilo moguće, neophodno je raditi backup sistema na određen period. Kada se pojavi greška visokog prioriteta, testeri su u obavezi da rade isključivo na toj grešci kako bi ona bila rešena. To znači da je potrebno obavestiti programere o kritičnoj grešci I sačekati da oni završe sa ispravkom nakon čega će ponovo uraditi isti testi I samo u slučaju da je ponovljen test prošao, može se nastaviti sa radom. Isto ovo rešenje se primenjuje I u slučaju greške visokog prioriteta.

7.8 Vremenski raspored

S obzirom na to da ovaj projekat ima određen termin za isporuku, neophodno je napraviti struktuisan vremenski raspored koji će prikazivati početni i krajnji datum izvršenja svake akcije koja je pripisana timu. Vremenskim rasporedom se bavi menadžer projekta koji će isti dostaviti članovima tima kako bi imali uvid u vremenski interval koji im je dat za izradu svog dela projekta. Na taj način se izrada sistema organizuje i sa vremenskog aspekta. Kritične situacije mogu da nastanu kada se vremenski period zbog neočekivanih situacija promeni. U tom slučaju je potrebno suziti vreme za izradu drugih akcija ili obavestiti stakeholdere da sistem kasni sa isporukom. Jako je bitno izbegavati drugi način baš iz razloga što će stakeholderi izgubiti poverenje. U tabeli u nastavku testu biće prikazane faze razvoja softvera, aktivnosti u okviru njih, kao i vreme potrebno za njihovu realizaciju.

Faza	Broj iteracije	Početak	Kraj
Početak	1	22.1.2021	20.2.2021
Pisanje dokumentacije zahteva	2	21.2.2021	30.3.2021
Implementacija	3	1.3.2021	1.4.2021

Sledeća tabela je malo detaljniji raspored aktivnosti kao I vremenskih intervala potrebnih za izvršenje pojedinih faza izrade celokupnog softvera.

ID	AKTIVNOST	POČETAK	KRAJ
1.	Analiza i dizajn	22.01.2021	20.02.2021
2.	Planiranje generalnog pristupa testiranju	21.02. 2021	30.02.2021
3.	Testiranje dizajna	1.03. 2021	11.03.2021

4.	Implementacija	12.03. 2021	01.04. 2021
5.	Dizajn test slučajeva	02.04. 2021	03.04. 2021
6.	Definisanje relacija među testovima	03.04. 2021	07.04. 2021
7.	Priprema stabova	08.04. 2021	18.04. 2021
8.	Testiranje	19.04. 2021	29.04. 2021

7.9 Pass/Fail testiranje-Black box

Prošao/Nije prošao test kriterijum se zasniva na ulazima i očekivanim izlazima iz sistema. Može se reći da spada u sistemsko testiranje, pošto obuhvata rad celog sistema da bi dobili izlazne podatke. Obuhvata testiranje funkcionalnih i nefunkcionalnih zahteva klijenata koji su definisani u SRS dokumentu, a čije je rešenje dato u arhitekturi sistema.

7.9.1 Pass testovi

U nastavku će biti dat pregled testova koji su prošli.

7.9.1.1 Prijava klijenata na sistem i unos lozinke

Naslov	Prijava klijenata	Rev1	Autor	Predrag Jevtić	Datum	23.03.2021
Cilj	Provera ispravnosti funkcije za prijavu		Reference	/		
Test uslovi	Instaliran sistem	Vreme neophodno za izradu test slucaja	10min	Vreme neophodno za izvrsenje test slucaja	10min	

	Opis postavke za testiranje
*	Sistem je trenutno postavljen u stanje u kome zahteva unos lozinke od klijenata
*	Klijent je u obavezi da unese ispravnu lozinku ukoliko želi da se prijavi
*	Klijent pristupa unosu lozinke

Definicija testa				Izvrsenje testa	
	Uslovi	Ulazni podaci	Ocekivani rezultati	Aktuelni rezultati	Broj problema
*	Potrebno je uneti ispravnu lozinu	Unosimo pogrešnu lozinku radi provere bezbednosti sistema	Obaveštenje o grešci	Sistem nas obaveštava da uneti podaci nisu ispravni i traži od klijenta ponovni unos podataka	/

	Opis postuslova
*	Nakon ponovnog unosa ispravne lozinke, korisnik je bez problema pristupio sistemu

7.9.1.2 Izbor senzor mod-a

Naslov	Izbor senzor mod-a	Rev1	Autor	Predrag Jevtić	Datum	23.03.2021.
Cilj	Provera dostupnosti senzor mod-a		Reference	/		
Test uslovi	Instaliran sistem	Vreme neophodno za izradu test slucaja	10min	Vreme neophodno za izvrsenje test slucaja	10min	

	Opis postavke za testiranje
*	Neophodno je da klijent bude prijavljen na sistem prethodno unetom pravilnom lozinkom
*	Klijent bira opciju za pristup senzor mod-u

Definicija testa				Izvrsenje testa	
	Uslovi	Ulazni podaci	Ocekivani rezultati	Aktuelni rezultati	Broj problema
*	Korisnik treba da izabere opciju za pristup senzor mod-u	Pritisak prsta na touchscreen display	Prikaz opcija u otvorenom senzor mod-u	Korisniku je prikazan senzor mod	

	Opis postuslova
*	Korisnik je uspešno pristupio senzor mod-u
*	Test uspešno prošao

7.9.1.3 Rad sa senzorima za toplotu

Naslov	Rad sa senzorima za toplotu	Rev1	Autor	Predrag Jevtić	Datum	23.03.2021.
Cilj	Podešavanje temperature za aktivaciju alarma		Reference	/		
Test uslovi	Instaliran sistem	Vreme neophodno za izradu test slucaja	20min	Vreme neophodno za izvrsenje test slucaja	20min	

	Opis postavke za testiranje
*	Korisnik treba da ima pristup senzor mod-u
*	Korisnik bira konfiguraciju senzora za toplotu
*	Korisnik unosi željenu temperaturu na koju hoće da se sistem aktivira
*	Korisnik potvrđuje unos

Definicija testa				Izvrsenje testa	
	Uslovi	Ulazni podaci	Ocekivani rezultati	Aktuelni rezultati	Broj problema
*	Pristup konfiguraciji senzora za temperaturu	Unos preniske temperature radi provere ispravnosti sistema	Obaveštenje sistema da je uneta temperatura nemoguća za prihvatanje	Sistem daje obaveštenje korisniku da neće doći do aktivacije sistema jer je uneta temperatura preniska. Sistem traži od korisnika unos druge vrednosti	/

	Opis postuslova
*	Korisnik podešava temperaturu na 50C , kao krajnju granicu
*	Sistem prihvata unos

7.9.1.4 Rad sa magnetnim senzorima

Naslov	Rad sa magnetnim senzorima	Rev1	Autor	Predrag Jevtić	Datum	23.03.2021
Cilj	Podešavanje aktivacije senzora		Reference	/		
	nakon izlaska korisnika iz kuće					
Test uslovi	Instaliran sistem	Vreme neophodno za izradu test slucaja	15min	Vreme neophodno za izvrsenje test slucaja	15min	

	Opis postavke za testiranje
*	Korisnik treba da ima pristup senzor mod-u
*	Korisnik bira konfiguraciju magnetnih senzora
*	Korisnik bira opciju da se senzori aktiviraju nakon njegovog izlaska iz kuće
*	Korisnik potvrđuje opciju

Definicija testa				Izvrsenje testa	
	Uslovi	Ulazni podaci	Ocekivani rezultati	Aktuelni rezultati	Broj problema
*	Ulazna vrata treba da budu zatvorena. Nakon aktiviranja ove opcije senzor će detektovati jedno otvaranje i jedno zatvaranje vrata (izlazak iz kuće) i nakon toga će se aktivirati i oglašavati alaramom prilikom svakog neindefikovanog otvaranja	Jednostavno podešavanje na touchscreen displey-u	Prihvatanje podešavanja	Sistem obaveštava korisnika da je podešavanje prihvaćeno i nudi mu opciju za obaveštavanje na povezanom uređaju	/

	Opis postuslova
*	Korisnik je uspešno podesio sistem za detektovanje provale kada je odsutan

7.9.1.5 Izbor kamera mod-a

Naslov	Izbor kamera mod-a	Rev1	Autor	Predrag Jevtić	Datum	23.03.2021
Cilj	Provera dostupnosti		Reference	/		
	kamera mod-a					
Test uslovi	Instaliran sistem	Vreme neophodno za izradu test slucaja	5m	Vreme neophodno za izvrsenje test slucaja	5m	

	Opis postavke za testiranje
*	Korisnik mora da bude prijavljen na sistem
*	Korisnik bira opciju za pristup kamera mod-u

Definicija testa				Izvrsenje testa	
	Uslovi	Ulazni podaci	Ocekivani rezultati	Aktuelni rezultati	Broj problema
*	Korisnik treba da izabere opciju za pristup kamera mod-u	Pritisak prsta na touchscreen display	Prikaz opcija u otvorenom kamera mod-u	Korisniku je prikazan kamera mod	

	Opis postuslova
*	Korisnik je uspešno pristupio kamera mod-u
*	Test uspešno prošao

7.9.1.6 Rad sa kamerom

Naslov	Rad sa kamerom	Rev1	Autor	Predrag Jevtić	Datum	23.03.2021.
Cilj	Prikaz snimaka kamera postavljenih oko kuće		Reference	/		
Test uslovi	Instaliran sistem	Vreme neophodno za izradu test slucaja	10m	Vreme neophodno za izvrsenje test slucaja	10m	

	Opis postavke za testiranje
*	Korisnik treba da ima pristup kamera mod-u
*	Korisnik treba da izabere prikaz snimaka

Definicija testa				Izvrsenje testa	
	Uslovi	Ulazni podaci	Ocekivani rezultati	Aktuelni rezultati	Broj problema
*	Korisnik bira opciju za prikaz snimaka	Unos komande na touchscreen displey-u	Prikaz snimaka	Sistem prikazuje snimke na displayu	/

	Opis postuslova
*	Korisnik je dobio uvid u snimke koje je kamera zabeležila

7.9.1.7 Rad sa senzorima za dim

Naslov	Rad sa senzorima za dim	Rev1	Autor	Predrag Jevtić	Datum	23.03.2021
Cilj	Provera rada konfiguracije koja se odnosi na rad ovih senzora		Reference	/		
Test uslovi	Instaliran sistem	Vreme neophodno za izradu test slucaja	10m	Vreme neophodno za izvrsenje test slucaja	10m	

	Opis postavke za testiranje
*	Potrebno je da korisnik bude prijavljen na sistem
*	Korisnik treba da pristupi senzor mod-u
*	Korisnik treba da izabere konfiguraciju senzora za dim

Definicija testa				Izvrsenje testa	
	Uslovi	Ulazni podaci	Ocekivani rezultati	Aktuelni rezultati	Broj problema
*	Korsnik sistema želi da isključi senzore za dim koji se nalaze u kuhinji pošto je trenutno u kući, želi da koristi šporet i ne želi aktiviranje alarma u takvoj situaciji	Ulazni podaci preko touchsreen display-a, koji se odnose na isključivanje ovih senzora u kuhinji, a u ostatku kuće na uključivanje	Obaveštenje korisniku da je konfiguracija uspešna	Sistem obaveštava korisnika da je operacija uspešno obavljena.	/

	Opis postuslova
*	Test uspešno prošao

7.9.1.8 Rad sa senzorima za pokret

Naslov	Rad sa senzorima za pokret	Rev1	Autor	Predrag Jevtić	Datum	23.03.2021.
Cilj	Korisnik želi da aktivira senzore koji identifikuju pokrete dok je vlasnik odsutan		Reference	/		
Test uslovi	Instaliran sistem	Vreme neophodno za izradu test slucaja	15m	Vreme neophodno za izvrsenje test slucaja	15m	

	Opis postavke za testiranje
*	Korisnik mora da bude prijavljen na sistem
*	Korisnik pristupa senzor mod-u
*	Korisnik bira konfiguraciju senzora za pokret

Definicija testa				Izvrsenje testa	
	Uslovi	Ulazni podaci	Ocekivani rezultati	Aktuelni rezultati	Broj problema
*	Senzori za pokret se aktiviraju kada vlasnik nije u kući, identifikuju kretanje mogućeg provalnika i sistem se oglašava alarmom	Ulazna podešavanja preko touchscreena. Korisnik podešava aktiviranje senzora nakon njegovog izlaska iz kuće	Prihvatanje podešavanja	Sistem obaveštava korisnika da će se senzori aktivirati nakon njegovog izlaska iz kuće	/

	Opis postuslova
*	Test uspešno prošao

7.9.2 Fail testovi

U nastavku će biti dat pregled testova koji nisu prošli

7.9.2.1 Provera sprečavanja neograničenog pokušaja logovanja

Naslov	Provera sprečavanja nasilnog logovanja	Rev1	Autor	Predrag Jevtić	Datum	3.4.2021
Cilj	Sprečavanje unosa lozinke neograničen broj puta		Reference	/		
Test uslovi	Instaliran sistem	Vreme neophodno za izradu test slucaja	15m	Vreme neophodno za izvrsenje test slucaja	15m	

	Opis postavke za testiranje
*	Korisnik sistema treba namerno da unese pogrešnu lozinku i potvrdi svoj unos
*	Nakon obaveštenja sistema da su podaci pogrešni, uneti ponovo pogrešnu lozinku i potvrditi unos

Definicija testa				Izvrsenje testa	
	Uslovi	Ulazni podaci	Ocekivani rezultati	Aktuelni rezultati	Broj problema
*	Korisnik treba da unosi pogrešnu lozinku i potvrđuje svoje unose dok god je to moguće	Pogrešni podaci	Nakon 3 puta pogrešno unete lozinke, sistem treba da obavesti korisnika da je ponovni unos moguć tek za određeno vreme	Korisnik i nakon trećeg pogrešnog unosa dobija mogućnost da odmah ponovo unese lozinku	1

	Opis postuslova
*	Test nije prošao

7.9.2.2 Pokretanje dodatnih funkcija senzora za pokret

Naslov	Pokretanje dodatnih funkcija senzora za pokret	Rev1	Autor	Predrag Jevtić	Datum	3.4.2021
Cilj	Provera funkcije za prepoznavanje kućnih ljubimaca		Reference	/		
Test uslovi	Instaliran sistem	Vreme neophodno za izradu test slucaja	10m	Vreme neophodno za izvrsenje test slucaja	10m	

	Opis postavke za testiranje
*	Nakon već objašnjene konfiguracije senzora za pokret, korisnik bira opciju za prepoznavanje kućnih ljubimaca
*	Korisnik potvrđuje unos

Definicija testa				Izvrsenje testa	
	Uslovi	Ulazni podaci	Ocekivani rezultati	Aktuelni rezultati	Broj problema
*	Korisnik treba da ima pristup konfiguraciji senzora za pokret. Nakon toga treba da ima mogućnost uključivanja funkcije za prepoznavanje kućnih ljubimaca, koja služi da se sistem ne oglašava ukoliko prepozna takav oblik	Čekiranje dodatne funkcije	Obaveštenje da je dodatna funkcija uključena	Sistem izbacuje obaveštenje da funkcija nije dostupna	1
	Opis postuslova				
*	Test nije prošao				

7.9.2.3 Podela ekrana display-a prilikom live prikaza kamere

Naslov	Podela ekrana display-a prilikom live prikaza kamere	Rev1	Autor	Predrag Jevtić	Datum	3.4.2021.
Cilj	Prikazivanje svih kamera na ekranu po principu border layout-a		Reference	/		
Test uslovi	Instaliran sistem	Vreme neophodno za izradu test slucaja	10m	Vreme neophodno za izvrsenje test slucaja	10m	

	Opis postavke za testiranje
*	Korisnik treba da ima pristup mod-u za nadzor
*	Korisnik treba omogući live prikaz kamera raspoređenih oko kuće

Definicija testa				Izvrsenje testa	
	Uslovi	Ulazni podaci	Ocekivani rezultati	Aktuelni rezultati	Broj problema
*	Pristup mod-u za nadzor	Zahtev za prikaz svih live prikaza kamere	Podela ekrana display-a na 4 dela (pošto je trenutno u sistemu 4 kamere) i prikaz live snimaka	Prikazivanje live snimaka jedan ispod drugog, po principu slide ekrana (skrolovanje)	1

	Opis postuslova
*	Test nije prošao

7.10 Jedinično testiranje-White box

Kao što je već navedeno u tački 6.4 ovakav vid testiranja odnosi se na sam kod. Ovo testiranje se obično vrši nakon kompletiranja sistema, kada imamo uvid u ceo kod sistema, tako da će ovde biti istestiran samo jedan mali deo sistema, pošto celokupna implementacija još uvek ne postoji.

Testiraću konfiguraciju senzora za temperaturu, i način na koji sistem radi ukoliko se unese preniska temperatura.

Najpe je dat deo koda koji se odnosi na metodu kojom se podešava temperatura:

```
public interface PodesavanjeTemeprature {

    public void temperatura(int temp);

}
```

Nakon toga, imamo deo koda koji se odnosi na samu konfiguraciju, i dodavanje podešenih senzora u listu koja je deo baze podataka predstavljene u delu o arhitekturi.

```
public class Konfiguracija implements PodesavanjeTemeprature{

    private List<PodesavanjeTemeprature> pt = new ArrayList();


    @Override
    public void temperatura(int temp) {
        for (PodesavanjeTemeprature podtemp : pt) {
            podtemp.temperatura(temp);
        }

    }

     public void add(PodesavanjeTemeprature podtemp) {
        this.pt.add(podtemp);
    }

}
```

Zatim imamo konkretnu klasu koja radi sa senzorima sa temperaturu. Ova klasa za početak čuva podatke o sobnoj temperaturi, što nam je jako bitno za nastavak rada. Ako sagledamo stvari logično, i podesimo temperaturu za aktivaciju koja je ista kao i u sobi, sistem će se istog trenutka oglasiti alarmom. Zato je potrebno sprečiti ovakav vid manipulacije sisemom. Ono što je još bitno, sobna temperatura može da varira (stepen-dva , gore-dole), ili u kuhinji može da bude par stepeni više ukoliko je šporet uključen. Zato je najniža temperatura za koju se sistem aktivira podešena na sobnaTemperatura + 5, što je i dato u sledećem delu koda.

Ukoliko je temperatura koja se unosi u konfiguraciju niža od dozvoljene, sistem sprečava takav unos:

```
public class TempSenzor implements PodesavanjeTemeprature{

    private int sobnTemperatura = 25;

    @Override
    public void temperatura(int temp) {
        if (temp <= (sobnTemperatura + 5)) {
            System.out.println("Vase podesavanje ne moze biti prihvaceno, jer je temperatura preniska");
        }

        else {
            System.out.println("Vas senzor je podesen na: " + temp + " stepeni");
        }
    }
}
```

Zatim imamo klijent, odnosno main klasu koja obezbeđuje prethodno navedeno. Klijent jedan senzor podešava na nižu temperaturu od dozvoljene, a drugi senzor pravilno konfiguriše:

```
public class Main {

    public Main() {

        PodesavanjeTemeprature senzorDnevnaSoba = new TempSenzor();
        PodesavanjeTemeprature senzorKuhinja = new TempSenzor();

        Konfiguracija k = new Konfiguracija();
        Konfiguracija k2 = new Konfiguracija();

        k.add(senzorDnevnaSoba);
        k2.add(senzorKuhinja);
        k.temperatura(23);
        k2.temperatura(50);

    }
```

I sada, nakon potvrde konfiugarije, možemo videti da li naš sistem sprečava prethodno navedene rizike:

```
run:
Vase podesavanje ne moze biti prihvaceno, jer je temperatura preniska
Vas senzor je podesen na: 50 stepeni
BUILD SUCCESSFUL (total time: 0 seconds)
```

Vidimo da je za jedan senzor podešavanje prihvaćeno, dok za drugi nije, tako da je ovaj test uspešno prošao.

8. Metrika

Merenje veličine softvera je izuzetno značajna aktivnost u procesu razvoja softvera.
Merenjem softvera se dobijaju informacije o veličini sistema koji je potrebno izgraditi, što je izuzetno važna informacija u procenama vremena potrebnog za izradu, cene i planiranja razvoja i održavanja softvera.

Kao mera veličine softverskog proizvoda najčešće se koristri metod procene grubih funkcionalnih tačaka a na osnovu kojih se može proceniti veličina sooftvera izražena brojem linija izvornog koda i na osnovu toga proceniti obim održavanja datog softvera.

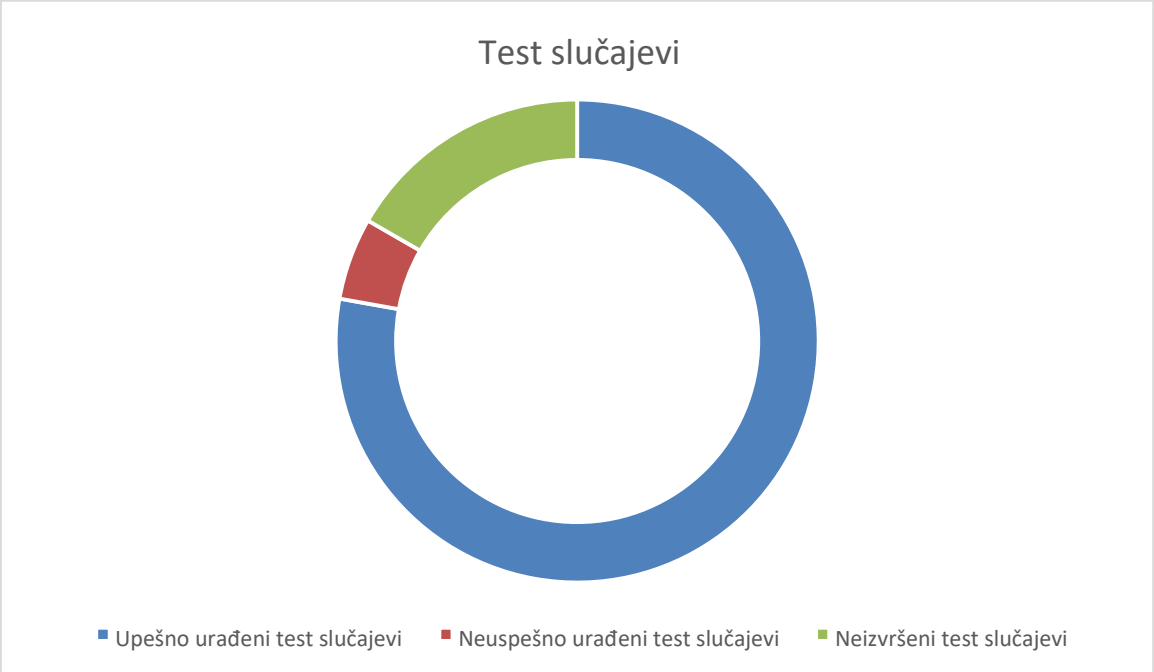
Metrika	Rezultat
Broj funkcionalnih zahteva	8
Broj test slučajeva po zahtevu	2
Broj test slučajeva za sve zahteve	16
Broj uspešno izvršenih test slučajeva	8
Broj izvršenih test slučajeva	11
Broj neuspešno izvršenih test slučajeva	3
Ukupan broj grešaka	Moguće je znati nakon izvršenja svih TS
Broj grešaka nižeg prioriteta	1
Broj grešaka srednjeg prioriteta	0
Broj grešaka visokog prioriteta	2

Iz tabele sada lako možemo izračunati procenat uspešnosti izvršenih test slučajeva:

procenat uspešnosti = 8/16 * 100 = 50%

Isto tako, možemo izračunati i procenat neuspešnosti:

procenat neuspešnosti = 3/16 * 100 = 18,75%



sl. Test slučajevi

8.1 Troškovi razvoja projekta

Pri kreiranju softvera, jako je bitno odrediti troškove razvoja istog. Pre svega treba odrediti satnicu svakog zaposlenog na razvoju softvera, a zatim i napraviti čitav plan razvoja projekta sa određenom gornjom i donjom granicom sati za svaku fazu razvoja softvera.

Sledeća tabela prikazuje procenu satnice za svakog zaposlenog:

Član tima	Cena po satu (u evrima)
Menadžer projekta	45
Menadžer kvaliteta	45
Programer	40
Dizajner interfejsa	35
Dizajner korisničkog iskustva	35
Tester	40

U tabeli ispod su prikazane faze razvoja softvera sa određenim vremenom rada:

Faza	Aktivnost	Minimalan broj sati	Maksimalan broj sati
Planiranje	Definisanje plana	20	30
	Određivanje vremena potrebnog da se projekat završi	2	4
	Procena budžeta	5	9
Definisanje zahteva	Prikupljanje zahteva	30	55
	Kreiranje SRS dokumenta	10	15
Dizajn	Prototip	3	5
Razvoj	Razvoj backenda	50	90
	Razvoj frontenda	50	90
	Dokumentovanje	10	15
Testiranje	Jedinični testovi	15	20
	Testiranje modula	20	30
	Testiranje integrisanog sistema	20	30
	Testiranje funkcionalnosti	30	50
	Testiranje GUI-a		
Održavanje	Uputsvo za instalaciju i upotrebu	10	20
	Ažuriranje verzija	5	10

Kada bi se sve ovo uzelo u obzir kao i skriveni troškovi, cena projekta bi iznosila oko 1000012000 evra.

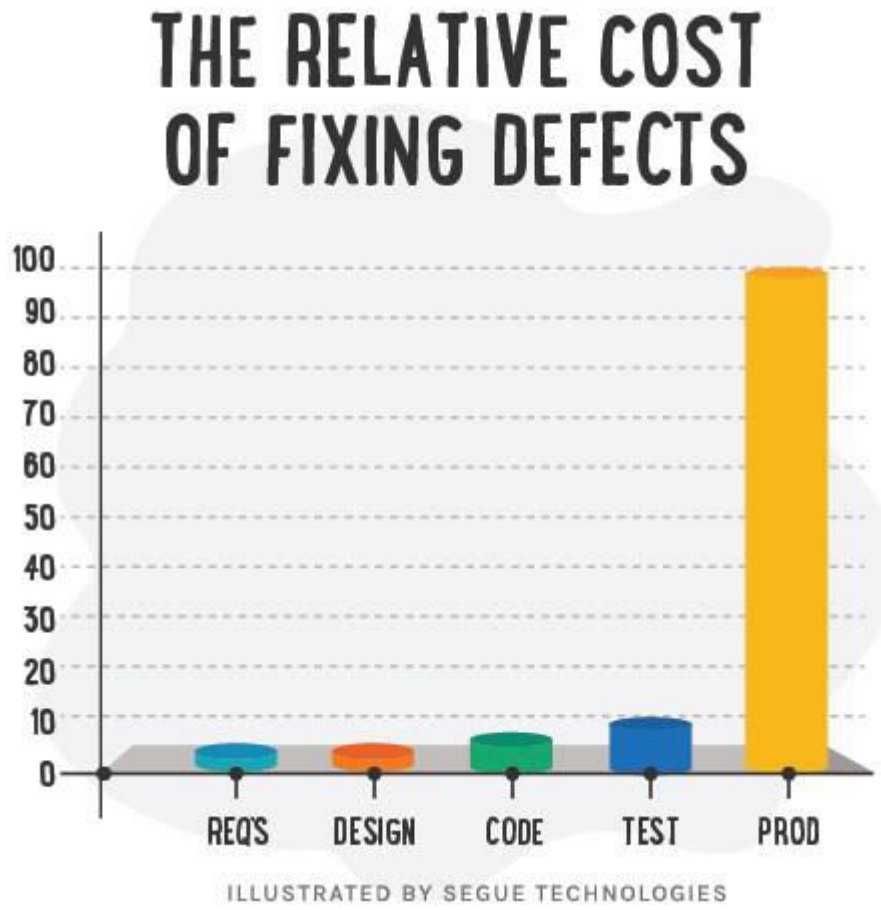
8.2 Troškovi uklanjanja defekata

Pregled i testiranje programa, kao aktivnosti osiguranja kvaliteta, služe kao filteri uklanjajući određeni postotak ulaznih defekata i ostavljajući deo neotkrivenih da budu otkriveni u sledećoj razvojnoj fazi. Troškovi uklanjanja defekata se izračunavaju za svaku aktivnost osiguranja kvaliteta množenjem broja uklonjenih nedostatka sa prosečnom cenom (troškom)

uklanjanja jednog defekta u toj fazi. Na tabeli ispod možemo videti cene troškova pri uklanjanju grešaka.

Br.	Faze razvoja softvera	Prosečna relativna cena otklanjanja greške (jedinica cene - €)
1.	Specifikacija zahteva	5
2.	Dizajn	10
3.	Kodiranje	30
4.	Testiranje	90
5.	Sistem nakon isporuke	1000

U aplikaciji trenutno postoje određeni defekti koji su dobijeni sistemskim testiranjem. U pitanju je jedna greška niskog i dve greške visokog prioriteta i potrebno je otkloniti ove probleme.



sl. Prikaz troškova uklanjanja defekata zavisno od faze[7]

9. Zaključak

Kroz prikazanu arhitekuru sistema jasno smo sagledali sve aspekte i delove celokupnog sistema i stekli jasnu sliku o načinu funkcionisanja istog. Prikazao sam način kako se sistem može razviti na dva tipa arhitekture, i dao jasne skice i dijagrame za taj deo sistema. “4+1” pogled nas je još detaljnije uveo u sve sekvence sistema, a na kraju smo sagledali i mogući način implementacije sistema. Kretanje kroz dokument i jasnu podelu na sekvence dala nam je Zahmanova matrica koja je u samom početku dokumenta.

Drugi deo dokumenta tiče se samog testiranja i metrike sistema. Istestirane su najbitnije funkcije samog sistema, dokumentovani rezultati i ukazano je na moguće greške koje mogu da se pojave. Istesitran je i jedan deo koda sistema, sa isečcima koda i objašnjenjima istog.

Na kraju je izražena procentualno uspešnost i neuspešnost testiranja slučajeva, kao i troškovi potrebni za implementaciju sistema i otklanjanja grešaka.

10. Literatura

1. Kuchana Partha, Software Architecture Design Patterns in Java (2004)
2. Dr. Dragan Domazet, MSc Nebojša Gavrilović, Projektovanje i Arhitektura softvera (2020/2021)
3. Dr. Svetlana Cvetanović, MSc Nebojša Gavrilović, Obezbeđenje kvaliteta, testiranje i održavanje softvera (2020/2021)
4. Zachman, John A., The ZF: The official Concise Definition of Zachman International Enterprise Archirtecture.
5. Sommerville, Software Engineering - Ninth Edition (2011)
6. The Economics of Software Quality – Capers Jones and Olivier Bonsignour
7. Quality Control: Critical to the Quality of Software Products – Segue Technologies, Inc.