Fenomenografska analiza percepcije studenata o ciljevima i iskustvima u učenju objektno orijentiranog programiranja

Šimunović, Dajana

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

DIPLOMSKI RAD

FENOMENOGRAFSKA ANALIZA PERCEPCIJE STUDENATA O CILJEVIMA I ISKUSTVIMA U UČENJU OBJEKTNO ORIJENTIRANOG PROGRAMIRANJA

Dajana Šimunović

Split, rujan 2024.

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

DIPLOMSKI RAD

FENOMENOGRAFSKA ANALIZA PERCEPCIJE STUDENATA O CILJEVIMA I ISKUSTVIMA U UČENJU OBJEKTNO ORIJENTIRANOG PROGRAMIRANJA

Dajana Šimunović

Split, rujan 2024.

Zahvala

Prije svega bih se zahvalila svojoj mentorici doc. dr. sc. Divni Krpan ali i svim ostalim profesorima od kojih sam imala prilike učiti i koji su me uspješno vodili do ovog posljednjeg trenutka studiranja.

Hvala najljepša!

Neizmjernu zahvalu na svemu dugujem i svojoj obitelji koja je uvijek bila podrška.

I na kraju bih se još zahvalila svojim prijateljima i prijateljicama, hvala vam što ste uvijek bili tu – za svaki razgovor, osmijeh, podršku i za sve trenutke koje smo proveli zajedno. Bez vas, ovih pet godina ne bi bilo ni upola toliko posebno i lijepo.

Temeljna dokumentacijska kartica

Sveučilište u Splitu, Prirodoslovno-matematički fakultet Diplomski rad

Odjel za informatiku Ruđera Boškovića 33, 21000 Split, Hrvatska

FENOMENOGRAFSKA ANALIZA PERCEPCIJE STUDENATA O CILJEVIMA I ISKUSTVIMA U UČENJU OBJEKTNO ORIJENTIRANOG PROGRAMIRANJA

Dajana Šimunović

SAŽETAK : Rad istražuje kako studenti percipiraju ciljeve i iskustva u učenju objektno orijentiranog programiranja koristeći fenomenografski pristup. Analizom intervjua i studentskih radova identificirani su različiti načini razumijevanja i pristupa OOP-u, što pruža uvid u njihove stavove, izazove i strategije učenja.

Ključne riječi : fenomenografija, objektno orijentirano programiranje, stav, učenje, metode

Rad sadrži : 36 stranica, 18 grafičkih prikaza i 18 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor : **Doc. dr. sc. Divna Krpan**, *docent Prirodoslovno matematičkog fakulteta Sveučilišta u Splitu*

Ocjenjivači: **Doc. dr. sc. Divna Krpan**, *docent Prirodoslovno matematičkog fakulteta Sveučilišta u Splitu*

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

Doc. dr. sc. Goran Zaharija, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen : *rujan 2024.*

Basic documentation card

University of Split, Faculty of science Master's Thesis

Department of Informatics Ruđera Boškovića 33, 21000 Split, Hrvatska

PHENOMENOGRAPHIC ANALYSIS OF STUDENTS' PERCEPTION OF THE GOALS AND EXPERIENCES IN LEARNING OBJECT ORIENTED PROGRAMMING

Dajana Šimunović

ABSTRACT : The paper explores how students perceive the goals and experiences of learning object-oriented programming using a phenomenographic approach. By analyzing interviews and student work, different ways of understanding and approaching OOP have been identified, providing insights into their attitudes, challenges, and learning strategies. The findings may help improve teaching methods and materials.

Keywords : phenomenography, object-oriented programming, attitude, learning, methods

Thesis consists of : 36 pages, 18 graphical representations, and 18 references. The original is in Croatian.

Mentor : **Divna Krpan**, *Ph.D Assistant Professor of Faculty of Science, University of Split*

Reviewers: **Divna Krpan**, *Ph.D Assistant Professor of Faculty of Science, University of Split*

Monika Mladenović, *Ph.D Assistant Professor of Faculty of Science, University of Split*

Goran Zaharija, *Ph.D. Assistant Professor of Faculty of Science, University of Split*

Thesis accepted : *September 2024.*

Sadržaj

Uvod

Objektno orijentirano programiranje (OOP) je jedan od najvažnijih i najšire korištenih paradigmi u suvremenom razvoju softvera. Njegovi ključni koncepti, poput objekata, klasa, nasljeđivanja, polimorfizma i enkapsulacije, čine temelj za izgradnju složenih, modularnih i održivih softverskih sustava. S obzirom na njegovu važnost, OOP je standardni dio kurikuluma informatičkih studija širom svijeta, uključujući i našu ustanovu. Međutim, kako studenti doživljavaju proces učenja OOP-a, koji su njihovi ciljevi i iskustva, te kako ta iskustva oblikuju njihov odnos prema programiranju, pitanja su koja zahtijevaju dublje razumijevanje. [1]

Fenomenografska analiza, kao kvalitativni istraživački pristup, pruža okvir za istraživanje raznolikosti percepcija među studentima. Dok fenomenologija istražuje individualna iskustva i subjektivne dojmove, fenomenografija se fokusira na varijacije u razumijevanju i doživljavanju fenomena među grupama ljudi. Fenomenografija istražuje varijacije u načinu na koji pojedinci percipiraju i doživljavaju određene fenomene, s fokusom na različite perspektive i razumijevanje ciljeva i koncepata unutar specifičnih edukacijskih okvira. [2] Ovaj pristup omogućava identifikaciju različitih načina na koje studenti percipiraju ciljeve i iskustva u učenju OOP-a, te kako te percepcije utječu na njihovo učenje i uspjeh. Ovim radom bi trebali saznati kako poboljšati učenje i pomnije razumijevanje glavnih elemenata objektno orijentiranog programiranja te motiva studenata za učenje ukazuje na motive koji mogu poboljšati učenje. Razvoj vještina kod studenata zahtijeva integraciju teorijskih koncepata s praktičnim iskustvom, a proces učenja često uključuje iterativne cikluse gdje studenti postupno usvajaju sve složenije koncepte kroz primjenu i refleksiju. [3] Dakle važno je naglasiti da razvoj vještina kod studenata nije linearan proces, već je on sam po sebi dinamičan i iterativan. Integracija teorijskih koncepata s praktičnim iskustvom predstavlja srž uspješnog učenja i omogućava studentima da stečena znanja primjenjuju u stvarnim situacijama u kojima će se naći. Ovaj pristup omogućava postupno usvajanje složenijih koncepata, gdje svaki korak učenja doprinosi dubljem razumijevanju i razvoju novih kompetencija kao i unaprjeđenje postojećih. Iterativni ciklusi učenja, koji uključuju primjenu, refleksiju i prilagodbu, ključni su za kontinuirani razvoj i usavršavanje vještina.

1. Usvajanje znanja o objektno orijentiranom programiranju

OOP predstavlja paradigmu programiranja koja se temelji na konceptu "objekata", koji mogu sadržavati podatke, u obliku polja, često nazvanih atributi; i kod, u obliku procedura, često nazvanih metode. Ova paradigma omogućava programerima da kreiraju module koji se mogu ponovno koristiti i koji se lako mogu modificirati bez utjecaja na druge dijelove programa. OOP je postao dominantan način pisanja softvera u zadnjih nekoliko desetljeća zbog svoje fleksibilnosti i efikasnosti u upravljanju kompleksnošću.

Za studente, razumijevanje OOP-a ne uključuje samo usvajanje teoretskih definicija, već i duboko razumijevanje i promišljanje o tome kako objekti i njihove međusobne interakcije mogu unaprijediti strukturu i organizaciju samog koda. Ključni koncepti OOP-a uključuju nasljeđivanje, enkapsulaciju i polimorfizam, koji omogućavaju kreiranje složenih i fleksibilnih sustava. Nasljeđivanje omogućuje stvaranje novih klasa temeljenih na postojećim klasama, čime se olakšava ponovna upotreba i proširivanje koda. Enkapsulacija omogućuje skrivanje unutarnjih detalja implementacije objekta, pružajući samo javne metode za interakciju s njim, što povećava sigurnost i smanjuje kompleksnost samog koda. Polimorfizam, s druge strane, omogućava objektnim metodama da se ponašaju različito ovisno o kontekstu situacije, čime se povećava fleksibilnost i prilagodljivost softverskih rješenja.

Stjecanje ovih koncepata kroz praksu i osobnu refleksiju pomaže studentima da razviju sposobnost kojom učinkovito upravljaju složenim projektima i omogućuje im da pišu kod koji je robusniji i zaista lakši za održavanje. Razumijevanje kako različite komponente OOP-a surađuju i utječu jedna na drugu ključno je za razvoj kvalitetnog softvera i uspješno upravljanje razvojnim procesima. OOP se ističe kao najprikladniji pristup za upravljanje kompleksnošću u razvoju softvera, pružajući studentima moćan alat za rješavanje izazova u profesionalnom programiranju. [4]

1.1. Prednosti OOP-a

OOP nudi niz prednosti koje su takav način programiranja učinile iznimno popularnim među programerima i široko primjenjivim u raznim dijelovima razvoja softvera. OOP omogućava ne samo bolju organizaciju koda, već i efikasnije upravljanje složenim sustavima kroz korištenje niza prednosti koje ima, a to su modularnost, ponovna upotrebljivost i lakše testiranje.

Jedna od ključnih prednosti OOP-a je modularnost. Kod se organizira u manje cjeline koje se nazivaju klase, pri čemu svaka klasa sadrži određeni skup podataka i metoda koji su specifični za jedan objekt. Ovakav pristup omogućava programerima da jasnije organiziraju kompleksne sustave u manje, jasnije i razumljivije dijelove. Modularnost ne samo da povećava čitljivost i održivost koda, već također omogućava paralelan razvoj različitih modula, što ubrzava razvojni proces. [4]

Ponovna upotrebljivost je također jedna ključna prednost OOP-a. Jednom definirane klase mogu se koristiti u različitim dijelovima programa ili čak u potpuno novim projektima, bez potrebe za ponovnim pisanjem istog koda. Ovaj aspekt omogućava programerima da izbjegnu ponavljanje koda, što smanjuje mogućnost samih grešaka i ubrzava razvoj aplikacija i programske podrške. Ponovna upotrebljivost koda značajno poboljšava efikasnost i smanjuje troškove održavanja softverskih sustava. [4]

OOP je posebno pogodno za upravljanje složenošću, što je ključna prednost prilikom rada na velikim i kompleksnim softverskim sustavima. Ovaj način modeliranja olakšava analizu i dizajn sustava, što čini OOP pogodnim za razvoj velikih sustava poput poslovnih aplikacija i baza podataka. [4] OOP omogućava programerima da modeliraju složene sustave kroz jednostavnije, međusobno povezane objekte. Ovi objekti mogu predstavljati stvarne entitete, kao što su korisnici, transakcije ili uređaji, čime se omogućava prirodnije razumijevanje i strukturiranje programa.

Jednostavnije testiranje i otklanjanje pogrešaka također je velika prednost OOP-a. Budući da su objekti i metode jasno definirani, programeri mogu lakše identificirati gdje dolazi do pogrešaka unutar sustava koji se gradi. Korištenje objekata omogućava izolaciju različitih dijelova programa, čime se lakše testiraju pojedini moduli bez utjecaja na druge dijelove koda. Dakle važna je jasna organizacija koda i objekata u smanjenju složenosti prilikom otklanjanja pogrešaka te pojednostavljivanju jednog procesa ispravljanja pogrešaka.

Primjena OOP-a u različitim industrijama dodatno potvrđuje njegovu fleksibilnost i sposobnost rješavanja kompleksnih problema.

1.2. Poučavanje objektno orijentiranog programiranja

Poučavanje OOP predstavlja važan i izazovan zadatak u obrazovanju budućih programera. Zbog svoje složenosti i apstraktnih koncepata, OOP zahtijeva od profesora da primijene pristupe koji studentima olakšavaju razumijevanje i usvajanje svih načela OOP-a. Efikasna metoda poučavanja OOP-a uključuje teorijsko objašnjavanje ključnih principa, praktičnu primjenu tih koncepata kroz zadatke i projekte, te korištenje vizualnih alata i dijagrama za bolje razumijevanje međuodnosa između objekata.

1.2.1. Teorijsko razumijevanje načela OOP-a

Prvi korak u poučavanju OOP-a je osiguravanje da studenti razumiju osnovne koncepte kao što su klase, objekti, nasljeđivanje, enkapsulacija i polimorfizam. Ovi pojmovi mogu biti apstraktni, pa je ključno pružiti jasne definicije i primjere iz stvarnog života koji pomažu studentima da povežu koncepte sa svakodnevnim situacijama. Klase se mogu smatrati nacrtima za stvaranje objekata, pri čemu objekti predstavljaju instance tih klasa. Klasa definira skup atributa (podataka) i metoda (funkcionalnosti) koje svi objekti te klase dijele. [4] Iznimno je važno jasno razložiti ove pojmove kroz jednostavne primjere prije nego se prijeđe na složenije aspekte OOP-a. Uspješno poučavanje OOP-a također uključuje prezentiranje primjera iz stvarnog života koji su relevantni za studente. Razvoj aplikacija koje simuliraju stvarne sustave, poput bankovnih aplikacija, upravljanja skladištem ili društvene mreža, omogućava studentima da shvate kako se OOP koristi u današnjoj industriji. Studenti se na ovaj način uče ne samo teoriji, već i kako primijeniti OOP koncepte za rješavanje stvarnih problema u industrijskim aplikacijama. Korištenje OOP-a za kreiranje sustava za upravljanje bazama podataka, gdje objekti predstavljaju stvarne entitete poput 'Transakcije' ili 'Korisnici', omogućuje studentima da primijene koncepte

poput enkapsulacije i nasljeđivanja. Ovaj primjer jasno prikazuje kako se OOP može koristiti za modeliranje složenih poslovnih sustava. [5]

1.2.2. Praktična primjena

Jedna od najvažnijih komponenti poučavanja OOP-a je omogućiti studentima da praktično primjene ono što su naučili. Pisanje koda kroz konkretne primjere i zadatke pomaže studentima da shvate kako se ti teorijski koncepti zapravo implementiraju u stvarnim programima. Projekti u kojima studenti moraju dizajnirati i razvijati vlastite klase, raditi s nasljeđivanjem, te koristiti polimorfizam u stvaranju složenijih aplikacija pružaju im dodatno iskustvo koje će kasnije naravno koristiti u profesionalnom razvoju. Praktičnu primjena je neizostavni dio poučavanja OOP-a. Kako bi profesori dodatno potaknuli studente na rad, oni mogu uključiti alate za vizualizaciju, poput UML dijagrama, koji pomažu studentima da bolje razumiju strukturu i odnose između objekata. Korištenje UML-a može značajno olakšati studentima da sa apstraktnih koncepata naprave lakši prijelaz na konkretne implementacije. [4] Efektivna metoda učenja OOP-a uključuje iterativne cikluse u kojima studenti kreiraju projekte, analiziraju svoj kod i prilagođavaju ga kako bi poboljšali implementaciju samoga koda. Ovaj pristup ne samo da studentima omogućuje da vježbaju pisanje koda, već i da kritički razmišljaju o dizajnerskim odlukama, optimizaciji i ponovnom poboljšavanju koda.

Paradigma objektno orijentiranog programiranja razvijena je za rješavanje većih problema koji zahtijevaju mnoge međusobno povezane klase. U podacima koje sam pronašao, najbolje učenje potiče se kada studenti moraju riješiti veće obvezne zadatke. Neki od njih uključivali su nekoliko klasa i stoga su bolje odražavali ideje iza paradigme u usporedbi s manjim zadacima. [9]

Studenti bi trebali biti poticani da uče kroz greške, uz stalno revidiranje svojih rješenja i dodatno poboljšavanje kroz povratne informacije instruktora i kolega. Iterativni razvoj kroz projekte, najbolji je način da studenti u potpunosti shvate kako OOP principi funkcioniraju u složenim aplikacijama. [4]

2. Poteškoće početnika u OOP-u

Početnici u OOP često se suočavaju s brojnim izazovima pri usvajanju osnovnih koncepata ove paradigme. OOP uvodi drugačiji način razmišljanja u odnosu na proceduralne ili funkcionalne paradigme, što donosi mnoge razlike u dosadašnjem načinu razmišljanja i kreiranju programskih rješenja. Iako smo nabrojali brojne prednosti OOP-a, početnici često imaju probleme s apstraktnim konceptima, složenom terminologijom i načelima te na samom kraju sa praktičnom primjenom tih koncepata. Sada ćemo razmotriti ključne poteškoće s kojima se početnici susreću u učenju OOP-a i kako te prepreke mogu utjecati na njihov napredak i samu motivaciju kod učenja programiranja.

2.1. Razumijevanje apstraktnih koncepata

Jedan od najvećih zamijećenih izazova za početnike u OOP-u je razumijevanje apstraktnih koncepata kao što su klase, objekti, nasljeđivanje, polimorfizam i enkapsulacija. Iako su ovi koncepti temeljni za OOP, mnogim studentima nedostaje intuitivno razumijevanje tj. stvarni primjer onoga što oni zapravo predstavljaju i kako se ponašaju. Na primjer, klase i objekti često se objašnjavaju kroz analogije s stvarnim svijetom, ali povezivanje tih pojmova sa stvarnom implementacijom koda može biti zbunjujuće i prema samim istraživanjima je to potvrđeno. Učenici često razumiju konceptualnu analogiju između stvarnog svijeta i OOP-a, ali imaju poteškoće u primjeni tih apstraktnih pojmova prilikom implementacije u kodu, osobito s nasljeđivanjem i apstrakcijom. Ovaj rad ukazuje na dugotrajne izazove u povezivanju teorije s praktičnom primjenom u programiranju. [6]

2.2. Razmišljanje u terminima objekata

Početnici koji dolaze iz proceduralnog programiranja često imaju poteškoća u prijelazu na objektno orijentirani način razmišljanja. U proceduralnom programiranju sam naglasak je na kreiranim funkcijama i podacima, dok je u OOP-u fokus na modeliranju stvarnih entiteta kroz novo naučene objekte i klase. Razumijevanje da su podaci i metode objedinjeni unutar objekata, te da sami objekti odrađuju komunikaciju putem metoda, može predstavljati izazov. Studenti mogu nastaviti razmišljati proceduralno, što otežava potpuno razumijevanje prednosti OOP-a, poput enkapsulacije i modularnosti. Promjena načina razmišljanja iz proceduralnog u objektno orijentirani pristup zahtijeva vrijeme i praksu jer zahtijeva drugačiji mentalni model. [7]

2.3. Nasljeđivanje i hijerarhija klasa

Koncept nasljeđivanja je posebno izazovan za početnike. Iako se nasljeđivanje može činiti jednostavnim jer nam omogućuje kreiranje novih klasa koje preuzimaju karakteristike onih već postojećih, studenti često ne razumiju kada i zašto bi trebali koristiti nasljeđivanje. Problemi se javljaju i pri razumijevanju dubokih hijerarhija klasa, gdje višestruki slojevi nasljeđivanja mogu postati teži za razumjeti . Osim toga, studenti često imaju problema s razlikovanjem između jednostavne i višestruke upotrebe nasljeđivanja, te sa situacijama u kojima nasljeđivanje treba izbjegavati. Učenje nasljeđivanja je složeno jer uključuje razumijevanje odnosa među samim klasama i apstraktnim klasama, što može otežati usvajanje ovog koncepta kod početnika koji trebaju vremena za usvojiti iste. [2]

2.4. Upravljanje složenošću

Početnici se također bore s upravljanjem složenošću softverskih rješenja u OOP-u. Iako OOP nudi alate za smanjenje složenosti kroz modularnost i ponovno korištenje već postojećeg koda, dizajniranje sustava koji koristi te značajke može biti izniman izazov

stvoriti poteškoće za početnike. Strukture koje uključuju više klasa i objekata mogu postati zbunjujuće, posebno kada se integriraju različiti OOP koncepti kao što su nasljeđivanje, enkapsulacija i polimorfizam. Ovo može rezultirati frustracijom kod studenata, koji često ne vide neposredne koristi OOP-a zbog njegove početne složenosti. Stoga, iako OOP omogućava lakše upravljanje složenim sustavima, studenti na početku često imaju poteškoća s organizacijom koda na način koji koristi prednosti OOP-a. [8] Da bi se ublažile ove poteškoće, važno je pružiti studentima jasne i razumljive smjernice, kao i konkretne primjere koji ilustriraju kako pravilno primijeniti OOP principe. Postavljanje jednostavnih, ali sveobuhvatnih zadataka, zajedno s realnim primjerima iz stvarnog svijeta, može pomoći studentima da bolje razumiju kako koristiti OOP za izgradnju organiziranih i održivih softverskih rješenja.

2.5. Poteškoće s primjenom teorije u praksi

Iako početnici s vremenom mogu razumjeti i počinju shvaćati teorijske koncepte OOP-a, kao što su klase, objekti i nasljeđivanje, često im nedostaje sposobnost da te iste koncepte primijene u stvarnim situacijama. Prelazak sa apstraktnih definicija na pisanje stvarnog koda, koji koristi te koncepte, može biti izazovan. Početnici često ne znaju kako strukturirati svoj kod koristeći OOP principe na način koji je skalabilan i održiv te uporabljiv.

Jedan od ključnih izazova je prijevod apstraktnih koncepata u konkretan kod kojeg oni sami trebaju napisati. Na primjer, studenti mogu razumjeti teorijski koncept nasljeđivanja kao način stvaranja novih klasa temeljenih na postojećim klasama, ali mogu imati problema u praktičnoj primjeni tog istog koncepta kada treba dizajnirati malo složeniji sustav. Bez konkretnih smjernica i primjera profesora, studenti mogu pretjerati s nasljeđivanjem, što može dovesti do složenih i teško održivih hijerarhija klasa, ili ga pak previše ograničiti, ne koristeći prednosti koje pruža sam princip OOP-a.

Također, strukturiranje koda prema OOP principima može biti teško zbog složenosti planiranja i dizajna, ako studenti nisu na to navikli. Na primjer, odlučivanje kako pravilno organizirati klase i njihove međusobne odnose zahtijeva duboko razumijevanje, promišljanje o problemu, razumijevanje dizajnerskih obrazaca i principa što može biti

zastrašujuće za početnike. Oni mogu imati poteškoća u prepoznavanju kada koristiti enkapsulaciju za zaštitu podataka ili kako implementirati polimorfizam za omogućavanje fleksibilnih i proširivih sustava koji će nam možda sutra trebati. Integracija OOP principa u postojeći kod također može predstavljati izazov. U stvarnim projektima, kod često nije pisan s OOP principima na umu, pa može biti teško rekonstruirati ili refaktorirati kod kako bi se uvele te potrebne značajke. Početnici mogu osjećati frustraciju kada pokušavaju prilagoditi postojeće rješenje OOP metodologiji, pogotovo ako kod nije dobro strukturiran ili ako ne postoje jasne granice između različitih funkcionalnih područja što je još izazovnije.

Osim toga, početnici se mogu suočiti s problemima u održavanju i proširivanju kodne baze. Kroz primjenu OOP-a, studenti bi trebali naučiti kako pisati kod koji je lako održavati i koji je moguće ovisno o budućim zahtjevima proširivati. Međutim, bez pravilnog razumijevanja dizajna klasa i objekata, kod koji na prvi pogled izgleda organizirano može postati teško organizirati s prolaskom vremena, što može otežati dodavanje novih značajki ili ispravke grešaka koje se zamijete.

Da bi se prevladali ovi izazovi, važno je osigurati studentima praktične primjere i usmjerene zadatke koji simuliraju takve stvarne scenarije koje će kad tad u budućem radu zateći. Rad na projektima koji zahtijevaju implementaciju OOP principa u stvarnom kodu može pomoći studentima da bolje razumiju kako ta naučena teorija prelazi u praksu tj. stvaran kod i program. Uvođenje povratnih informacija i mentorstva također može igrati ključnu ulogu u pomaganje studentima da poboljšaju svoje vještine u primjeni OOP-a.

3. Metodologija istraživanja

3.1. Predmet i cilj istraživanja

Predmet ovog istraživačkog rada je fenomenografska analiza percepcije studenata o ciljevima i iskustvima u učenju pri pohađanju kolegija OOP na Prirodoslovno-matematičkom fakultetu u Splitu. Fenomenografija je istraživački pristup koji se koristi za istraživanje različitih načina na koje ljudi doživljavaju i razumiju određene fenomene. Cilj je identificirati različite varijacije u percepciji i razumijevanju kako bi se dobio uvid u načine na koje studenti pristupaju učenju OOP-a.

Ciljevi ovog istraživanja su:

- Identificirati različite načine na koje studenti percipiraju ciljeve učenja OOP-a.
- Istražiti iskustva studenata tijekom procesa učenja OOP-a.
- Razumjeti kako različite percepcije i iskustva utječu na uspjeh u učenju OOP-a.
- Identificirati percepcije i iskustva studenata o učenju programiranja i programskih jezika.
- Razumjeti ciljeve studenata u učenju programiranja.
- Ispitati najzahtjevnije aspekte programiranja za studente.
- Istražiti percepcije studenata o ključnim konceptima OOP-a.
- Identificirati resurse koje studenti koriste za učenje OOP-a.

3.2. Instrumenti

U ovom istraživanju korišten je upitnik kao glavni instrument za prikupljanje podataka. Upitnik je bio dizajniran tako da obuhvati različite aspekte percepcije i iskustava studenata u vezi s učenjem OOP.

3.2.1. Struktura upitnika

Upitnik se sastoji od 12 pitanja koja se mogu podijeliti u dvije glavne kategorije: demografska i akademska pitanja, te pitanja o percepciji i iskustvima učenja OOP-a.

Demografska pitanja :

Spol: Pitanje je omogućilo tri opcije odgovora (M, Ž, Ne želim se izjasniti) kako bi se osigurala inkluzivnost.

Godina studija: Pitanje je uključivalo četiri opcije (2. preddiplomskog, 3. preddiplomskog, 1. diplomskog, 2. diplomskog) kako bi se identificirala razina studija.

Prvi put upisan kolegij OOP: Pitanje s opcijama Da i Ne radi identifikacije prethodnog iskustva s kolegijem.

Okvir korišten za učenje OOP: Pitanje je uključivalo opcije JavaScript i C#, što su najčešće korišteni programski okviri u nastavi.

Smjer studija: Pitanje je uključivalo tri opcije (Informatika, Informatika tehnika, Ostali) kako bi se identificirao smjer studija sudionika.

Pitanja o percepciji i iskustvima učenja OOP-a:

Objasnite kako doživljavate učenje programiranja i programskih jezika. Otvoreno pitanje koje omogućava studentima da izraze svoja osobna iskustva i percepcije.

Koji je cilj vašeg učenja programiranja?: Otvoreno pitanje s ciljem identificiranja motivacija i ciljeva studenata.

Što vam je programerski najzahtjevnije na studiju?: Otvoreno pitanje koje istražuje najzahtjevnije aspekte studija prema percepciji studenata.

Kako doživljavate pojam objekta u kontekstu objektno-orijentiranog programiranja. Pokušajte navesti što više ključnih riječi i izraza kojima bi opisali objekt. Otvoreno pitanje koje traži od studenata da navedu ključne riječi i izraze kojima opisuju objekt.

Kako doživljavate pojam klase u kontekstu objektno-orijentiranog programiranja . Pokušajte navesti što više ključnih riječi i izraza kojima bi opisali klasu. Otvoreno pitanje koje traži od studenata da navedu ključne riječi i izraze kojima opisuju klasu.

Objasnite koja je razlika između objekta i klase?: Otvoreno pitanje koje ispituje razumijevanje studenata o razlici između objekta i klase.

Koje ste resurse koristili za učenje OOP?: Otvoreno pitanje koje identificira resurse koje studenti koriste za učenje.

Sudionici su bili informirani o cilju istraživanja i osigurana im je povjerljivost njihovih odgovora. Upitnik je bio distribuiran u papirnatom obliku.

3.3. Sudionici

Sudionici ovog istraživanja studenti su prijediplomskih te diplomskih studija na Prirodoslovno matematičkom fakultetu u Splitu koji ujedno slušaju kolegij OOP . OOP je kolegij kojeg studenti polažu na 2. i 3. godini prijediplomskih studija te na 1. i 2. godini diplomskih studija. U ovom istraživanju sudjelovali su studenti prijediplomskih studija Informatika i tehnika, Informatika, Matematika, te Matematika i informatika i Fizika. Kako je statistički malen broj studenata u studijskoj grupi Matematika i informatika, Matematika i Fizika u svrhu ovog istraživanja ti smjerovi su označeni kao Ostali. Što se tiče spola, u istraživanju je sudjelovalo 24 studenta muškog i 25 studentica ženskog spola. 2 osobe se nisu htjele izjasniti po pitanju spola.

3.4. Postupak

Provedba ankete u okviru ovog istraživanja bila je temeljito planirana i izvedena kako bi se osigurala visokokvalitetna i pouzdana zbirka podataka. Proces je uključivao nekoliko ključnih koraka, od pripreme upitnika do analize prikupljenih podataka. Prvi korak u provedbi ankete bio je dizajn upitnika, koji je trebao biti sveobuhvatan, ali i dovoljno jasan i jednostavan za ispunjavanje. Upitnik je sadržavao 12 pitanja podijeljenih u dvije glavne kategorije:

Demografska i akademska pitanja: Ova pitanja su obuhvatila osnovne informacije o studentima, kao što su spol, godina studija, prethodno iskustvo s kolegijem OOP, okvir korišten za učenje OOP-a, i smjer studija.

Pitanja o percepciji i iskustvima učenja OOP-a: Ova pitanja su bila otvorenog tipa i omogućila su studentima da detaljno opišu svoja iskustva, ciljeve i izazove u učenju OOP-a.

Prije stvarne provedbe ankete, upitnik je prošao kroz pilot-testiranje s malim brojem studenata. Cilj pilot-testiranja bio je provjeriti jasnoću pitanja, razumljivost instrukcija, te procijeniti vrijeme potrebno za ispunjavanje upitnika. Na temelju povratnih informacija iz pilot-testiranja, napravljene su potrebne korekcije i prilagodbe upitnika.

Prije početka anketiranja, studenti su bili informirani o cilju i značaju istraživanja, kao i o postupku prikupljanja podataka. Istaknuto je da je sudjelovanje dobrovoljno i anonimno, te da se podaci koriste isključivo u istraživačke svrhe. Studenti su dobili upute kako pravilno ispuniti upitnik i osigurana im je pomoć u slučaju nejasnoća.

Upitnici su distribuirani tijekom redovnih sati nastave na kolegiju OOP u dogovoru s voditeljicom kolegija, kako bi se osigurao reprezentativni uzorak i visok odziv. Upitnici su bili distribuirani u papirnatom obliku. Svaki student je dobio jedan papir kako bi se osiguralo da svaki student ispunjava upitnik samo jednom.

Tijekom prikupljanja podataka, bila sam dostupna za pomoć i pojašnjenja, osiguravajući da studenti pravilno razumiju pitanja i da odgovori budu što je moguće precizniji. Studenti su imali dovoljno vremena za ispunjavanje upitnika kako bi mogli temeljito razmisliti o svojim odgovorima.

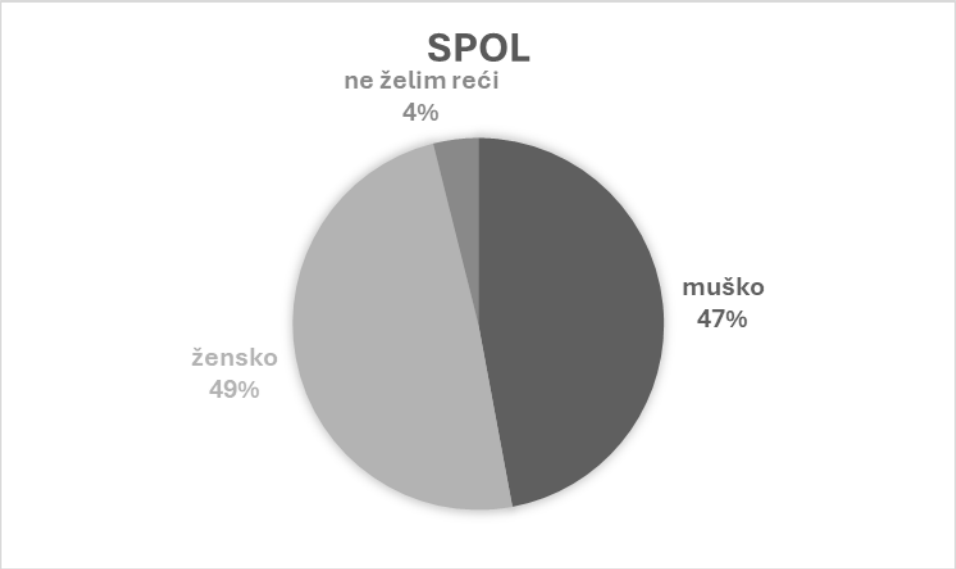
Nakon prikupljanja, svi papirnati upitnici su pažljivo pregledani i unijeti u SPSS program. Svi podaci su pohranjeni na siguran način, osiguravajući povjerljivost i anonimnost sudionika. Prikupljeni podaci su analizirani kvalitativnim i kvantitativnim metodama. Otvoreni odgovori su transkribirani i kodirani kako bi se identificirale ključne teme i varijacije u percepcijama i iskustvima studenata. Kategorički podaci su analizirani deskriptivnim statistikama kako bi se pružila osnovna demografska slika uzorka.

3.5. Rezultati istraživanja

U ovom poglavlju prikazani su rezultati deskriptivne statistike dobiveni iz odgovora na upitnik. Deskriptivna statistika pruža osnovni pregled prikupljenih podataka, uključujući frekvencije i proporcije odgovora za svaku kategoričku varijablu. Ova analiza pomaže u razumijevanju karakteristika uzorka sudionika i njihovih odgovora.

3.5.1. Spol

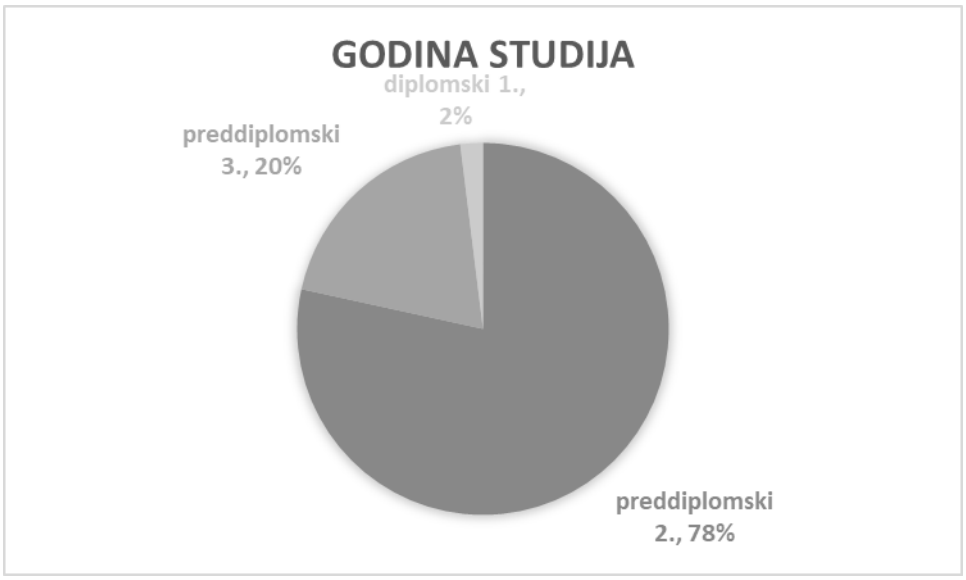
Većina sudionika bila je ženskog spola 25, dok je 24 bilo muškog spola. Manji postotak sudionika 2 nije želio izjasniti svoj spol.



Slika 1 Prikaz spola u postotcima

3.5.2. Godina studija

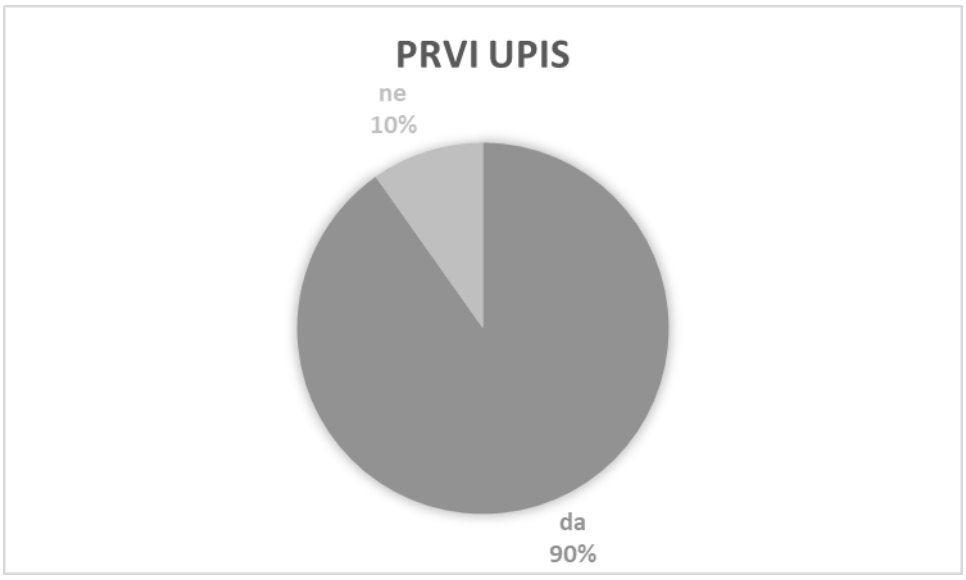
Najviše sudionika bilo je druga godina preddiplomskog studija (40), zatim treća godina preddiplomskog studija (10), dok je najmanje sudionika bilo prva godina diplomskog studija (1).



Slika 2 Prikaz godine studija u postotcima

3.5.3. Prvi upis

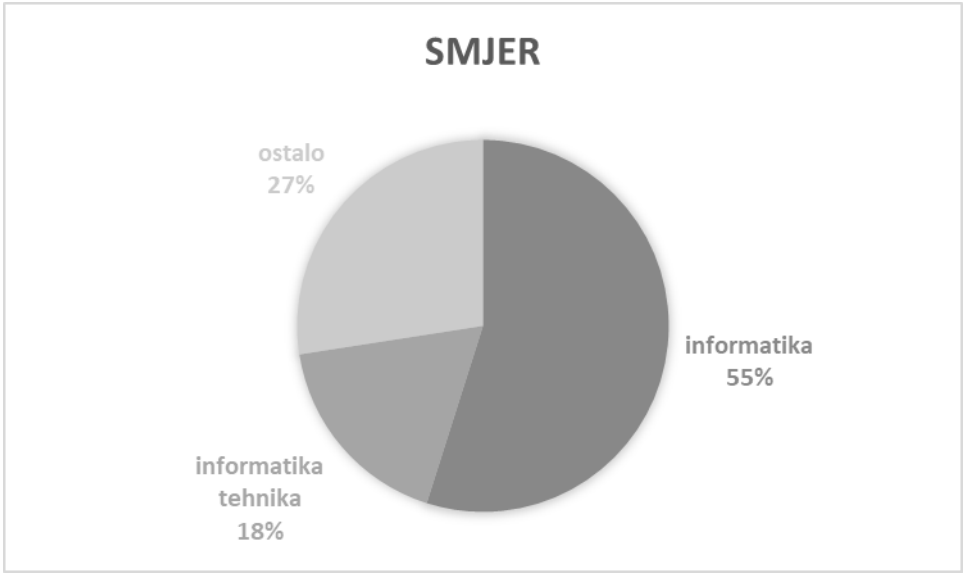
Većina sudionika (46) prvi put je upisala kolegij OOP, dok je 5 sudionika već prethodno upisalo ovaj kolegij.



Slika 3 Prikaz prvog upisa u postotcima

3.5.4. Smjer studijske grupe

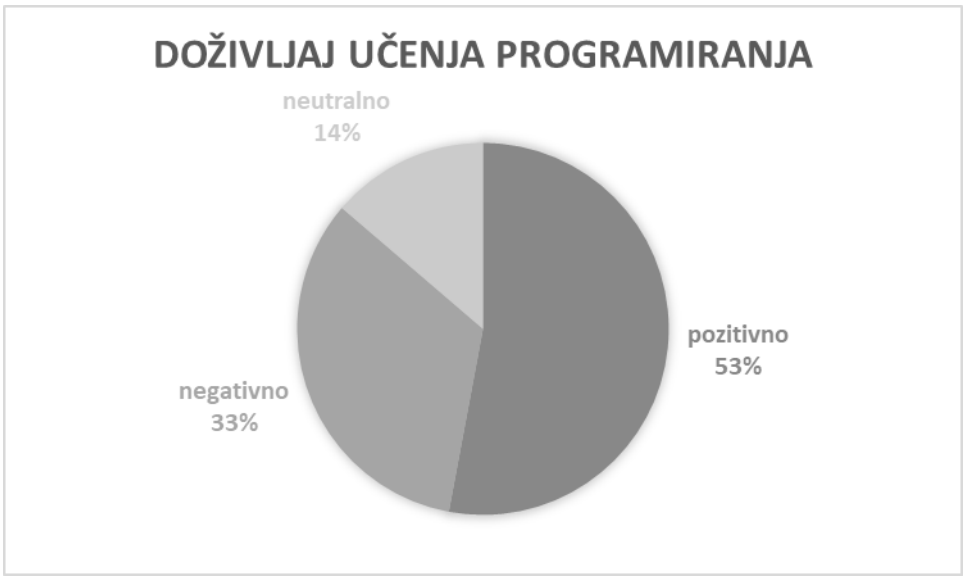
Najviše sudionika bilo je smjera Informatika (28), dok je 9 sudionika bilo smjera Informatika tehnika. Ostali smjerovi činili su 14 sudionika uzorka.



Slika 4 Prikaz smjera u postotcima

3.5.5. Doživljaj učenja programiranja

Većina studenata (27) doživljava učenje programiranja kao pozitivno iskustvo. Manji udio studenata (17) smatra učenje negativnim iskustvom, dok ih 7 ima neutralno mišljenje.



Slika 5 Prikaz doživljaja učenja programiranja u postotcima

Rezultati pokazuju da postoji raznolikost u doživljaju učenja programiranja među studentima. Dok većina studenata opisuje svoje iskustvo kao pozitivno ili zanimljivo, postoji značajan postotak studenata koji doživljavaju stres ili izražavaju frustraciju zbog izazova koje donosi učenje programiranja. Ovi nalazi sugeriraju potrebu za prilagodbom nastavnih metoda i strategija kako bi se potaknulo pozitivno shvaćanje učenja programiranja i smanjio stres ili frustracija kod studenata. Dodatna istraživanja mogla bi se fokusirati na identifikaciju specifičnih izazova koji doprinose negativnom doživljaju učenja programiranja te razvoj intervencija koje bi pomogle u prevladavanju tih izazova.

Analizirajući ključne riječi korištene u izjavama studenata o svojim iskustvima u učenju programiranja, primjetno je nekoliko različitih percepcija:

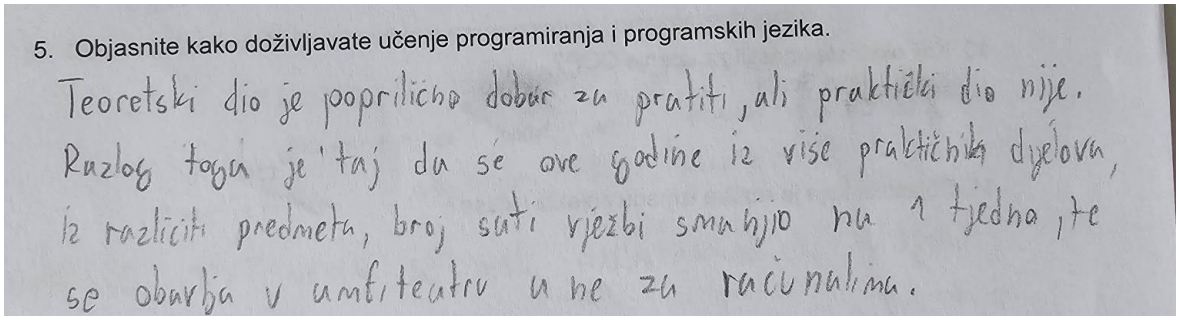
- Nema osjećaja napretka: Neki studenti osjećaju nedostatak napretka u svojem učenju, što može biti frustrirajuće i demotivirajuće.
- Dobro / Dosadno / Zanimljivo / Stresno / Zahtjevno: Postoje različite emocije i percepcije u vezi s procesom učenja programiranja. Dok neki studenti opisuju

iskustvo kao dobro ili zanimljivo, drugi ga doživljavaju kao dosadno ili stresno, što može ovisiti o njihovom osobnom interesu, predznanju ili iskustvu.

- Iznimno zanimljivo / Zanimljivo: Unatoč izazovima i stresu, neki studenti ističu da smatraju učenje programiranja iznimno zanimljivim ili jednostavno zanimljivim, što može ukazivati na njihovu strast prema programiranju ili na učinkovitost nastavnog pristupa.
- Loša organizacija: Postojanje percepcije loše organizacije sugerira da studenti možda imaju poteškoća s jasnoćom rasporeda, materijala ili uputa, što može ometati njihovo učenje.
- Želja za napretkom: Unatoč izazovima, neki studenti pokazuju motivaciju za napredovanjem i unaprjeđenjem svojih vještina u programiranju.

Ove različite percepcije odražavaju složenost učenja programiranja i naglašavaju važnost prilagođavanja nastavnog pristupa kako bi se potaknula motivacija i podržalo učenje kod svih studenata.

Na idućoj slici i ostalim navodima koji su navedeni iz anketnih odgovora možete vidjeti primjere odgovora koji potkrepljuju zaključak.



Slika 6 Odgovor na pitanje o doživljaju učenja programiranja

Također neki od odgovora su : „Ovisi kako koji predmet. Mislim da je nastava bila bolja dok smo imali vježbe po grupama od 2 sata nego ovako svi na jednom mjestu 45 min.“, „ Slabo. Nismo prezadovoljni. Učenje se svodi na napamet učenje nepotrebnih ili slabo potrebnih podataka.“, „Slabo nismo prezadovoljni, učenje se svodi na napamet učenje nepotrebnih ili slabo potrebnih podataka. Trenutno ne znamo programirati.“, „Nekad mi je

tlaka kada moram učiti, a ne razumijem što se događa, a onda mi prijateljica pojasni sve bude super i zanimljivo i baš uživam učiti onda i sve povezivati, ali na satu ništa ne razumijem.“, „Zamorno zbog bliceva i domaćih radova koji ponekad nemaju smisla. Također, satovi su umarajući jer ne naučim kodirati na taj način.“, „Jako zanimljiv i dobro osmišljen predmet.“, „Volim učiti programiranje, čini me sretnijim.“, „Doživljavam ga samo kao predmet koji moram položiti, za baviti sa željenim poslom koji ne uključuje programiranje.“, „Jako je zanimljivo rješavati svakodnevne probleme. Potiče kritičko razmišljanje, dobro je za razvoj kognitivnih sposobnosti.“ .

3.5.6. Cilj učenja programiranja

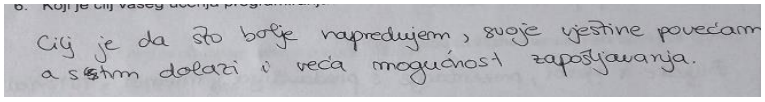
Najčešći cilj učenja programiranja među studentima je napredak (43). Isključivo položiti kolegij želi 3 studenta, dok 5 njih nema poseban cilj.

Slika 7 Prikaz cilja učenja programiranja u postotcima

Analizirajući ključne riječi korištene u izjavama studenata o svojim ciljevima u učenju programiranja, možemo primijetiti različite motivacije i svrhe:

- Napredak: Veliki broj studenata ističe napredak kao svoj cilj. To ukazuje na želju za poboljšanjem svojih vještina i znanja u programiranju.
- Posao: Značajan broj studenata vidi učenje programiranja kao sredstvo za stjecanje posla u području informacijske tehnologije, što može uključivati razne uloge u softverskom razvoju, analizi podataka, web dizajnu i slično.
- Isprogramirati svoju ideju: Postoji skupina studenata koji žele naučiti programiranje kako bi mogli ostvariti svoje vlastite ideje i projekte, možda kao poduzetnici ili kreatori softvera.
- Polaganje kolegija: Nešto manji broj studenata vidi cilj u učenju programiranja kako bi uspješno položili određeni kolegij ili ispit.
- Novac: Iako manje zastupljeno, neki studenti spominju novac kao motivaciju za učenje programiranja, vjerojatno misleći na potencijalno visoke plaće u IT sektoru.
- Naučiti osnove: Postoji i manji broj studenata koji ističu cilj učenja osnova programiranja, što može biti početna točka za daljnje napredovanje ili specijalizaciju.

Ovi ciljevi odražavaju raznolikost motiva studenata za učenje programiranja, od osobnog razvoja i kreativnosti do stjecanja konkretnih vještina za profesionalni uspjeh u IT industriji.

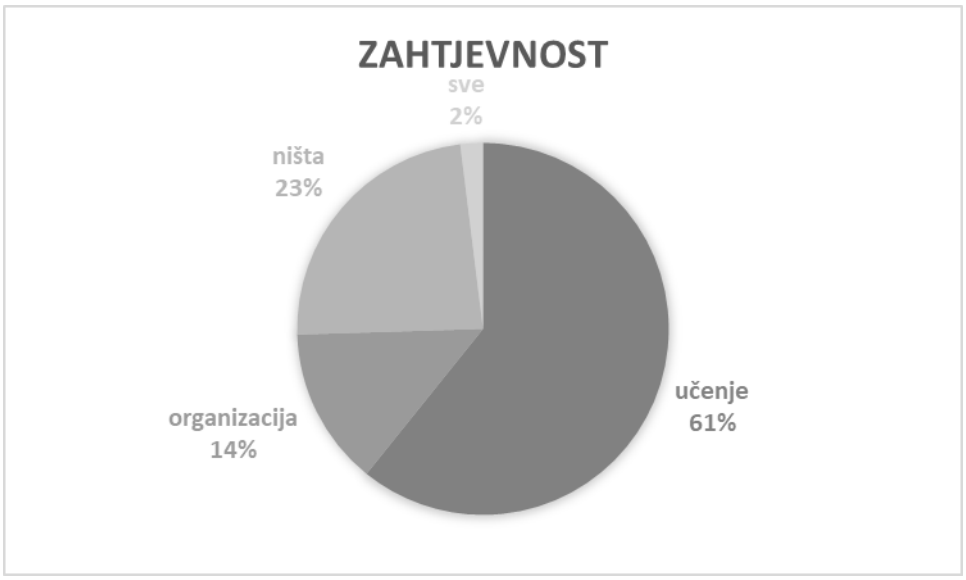


Slika 8 Odgovor na pitanje o cilju učenja programiranja

Neki od odgovora su i „Konstantno napredovanje.“, „Biti sposobna rješavati probleme problemske zadatke u više programskih jezika, izraditi nekakav projekt samostalno ili u timu.“, „Da naučim što više.“, „Isključivo polaganje kolegija, jer to nije područje kojim se želim baviti.“, „Cilj mi je steći što više znanja da samostalno mogu izvršiti svoje ideje te da lakše rješavam probleme.“, „Raditi ono što volim.“, „Pare.“, „Cilj je da što bolje napredujem,svoje vještine povećam, a s tim dolazi i veća mogućnost zapošljavanja.“, „Da obogatim svoje vještine, a i zabavno je i zanimljivo.“, „Uspješna karijera.“, „Pronaći predmet program koji mi se dovoljno sviđa da ga sam proučim u slobodno vrijeme.“.

3.5.7. Zahtjevnost

Najveći izazov za studente je učenje (31). Organizacija također predstavlja značajan izazov (7), dok sve smatra zahtjevno samo 1 student, a ništa njih čak 12.



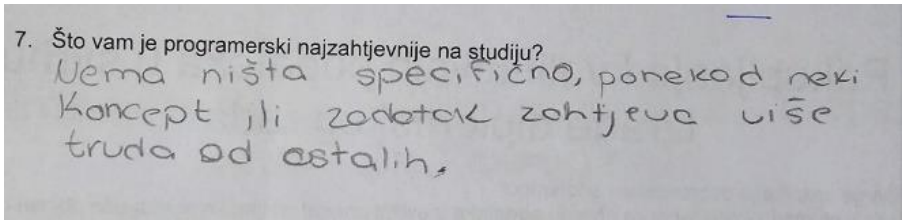
Slika 9 Prikaz zahtjevnosti u postotcima

Analizom ključnih riječi koje su studenti koristili u izjavama o zahtjevnosti učenja programiranja, otvara se uvid u različite aspekte koji predstavljaju izazove u ovom procesu.

- Studenti ističu izazove vezane uz praktičnu primjenu znanja kroz pisanje koda, što može biti složeno i zahtjevno, posebno za početnike.
- Ovaj izazov ukazuje na pritisak da se nauči velika količina materijala u ograničenom vremenskom periodu, što može uzrokovati stres i opterećenost.
- Studenti ističu nedovoljno iskustvo kao prepreku učenju programiranja, što može otežati razumijevanje i primjenu naučenih koncepata u praksi.
- Pronalaženje motivacije za učenje programiranja može biti izazovno, posebno kada se suočavaju s teškim ili apstraktnim konceptima.
- Proces otklanjanja grešaka može biti izazovan i zahtjeva vještine analitičkog razmišljanja i rješavanja problema.

- Studenti percipiraju projektni zadatak kao izazov koji zahtijeva primjenu svih naučenih koncepata u praksi, što može biti kompleksno i zahtjevno.
- Neki studenti ističu izazove vezane uz stvaranje animacija, što može zahtijevati dodatno znanje i vještine iz područja programiranja.

Svaki od ovih izazova predstavlja priliku za učenje i rast, ali istovremeno zahtijeva prilagođene strategije podrške kako bi se prevladali i osiguralo uspješno napredovanje studenata u učenju programiranja.

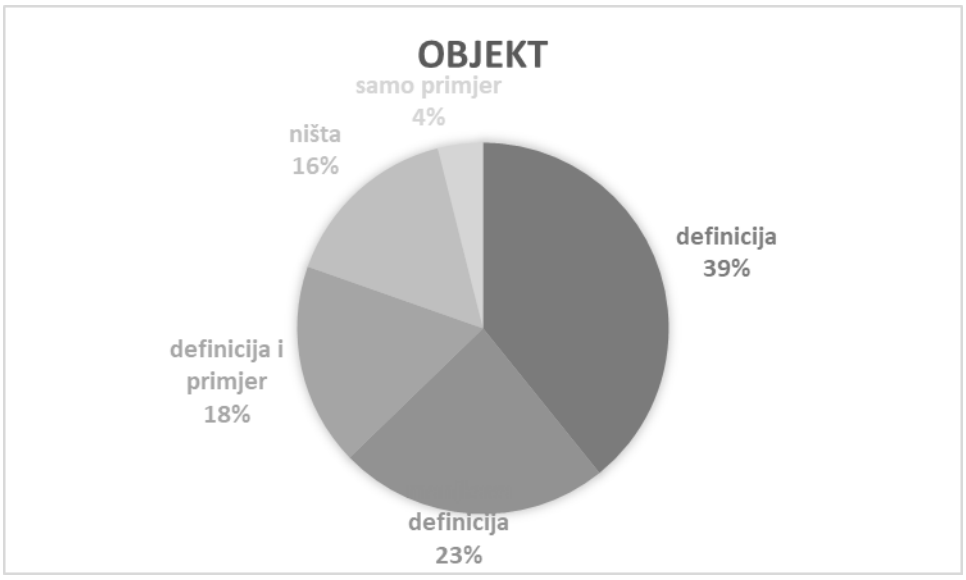


Slika 10 Primjer odgovora o zahtjevnosti

Također neki od odgovora su „Učenje teorije kada samo trebam pisati kod nije problem, ali kada krene teorija vezano s tim onda postane puno teže.“, „Nema ništa specifično, ponekad neki koncept ili zadatak zahtijeva više truda od ostalih.“, „Projektni zadatak “, „Sve mi je jednako teško.“, „Najzahtjevnije mi je naučiti programirati.“, „Nijedan aspekt nije posebno zahtjevan niti kompliciran. Razumijem koncepte bez problema.“, „Praćenje svih domaćih zadaća i projekata.“, „Pronaći rješenja za greške u kodu nakon što pokrenemo kod dobit ćemo grešku gdje puca kod no mjesto gdje se zapravo događa, greška je nepoznato.“.

3.5.8. Pojam objekta

Najčešće što studenti koriste za opisivanje objekta su definicija (20),a manjkavu definiciju koristi njih 12. Definiciju i primjer je upotrijebilo njih 9, samo primjer njih 2, a ništa nije napisalo njih 8.

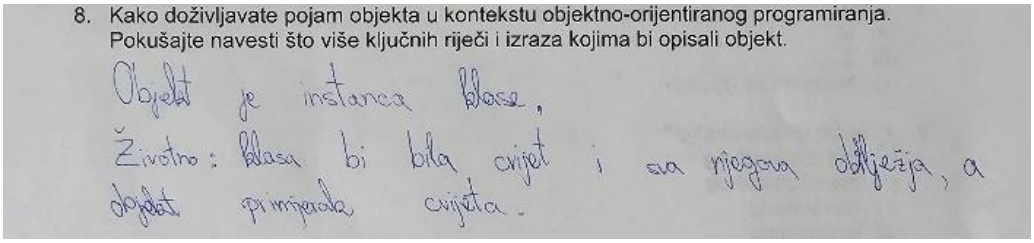


Slika 11 Prikaz izjava o objektu u postotcima

Analizom ključnih riječi koje su studenti koristili u izjavama o konceptu objekata u programiranju, možemo identificirati različite načine na koje percipiraju ovaj ključni koncept.

- Definicija: Za 39.2% studenata, naglasak je na razumijevanju jasne definicije objekta u programiranju. To ukazuje na važnost temeljnog razumijevanja konceptualnih osnova prije nego što se krene u primjenu.
- Manjkava definicija: 23.5% studenata izražava nedostatak jasne definicije objekta kao izazov koji ih sprječava u potpunom razumijevanju i primjeni ovog koncepta.
- Definicija i primjer: Za 17.6% studenata, kombinacija jasne definicije i primjera ključna je za razumijevanje koncepta objekata. Ovo sugerira da studenti preferiraju praktičan pristup uz teorijsko razumijevanje.
- Samo primjer: Manji postotak studenata (3.9%) ističe važnost primjera u razumijevanju objekata, bez naglaska na teorijsku definiciju.
- Ništa: Za 15.7% studenata, izražavanje da nisu imali jasno razumijevanje ili konceptualni okvir objekata u programiranju sugerira potrebu za dodatnim obrazovnim resursima ili podrškom.

Ova raznolikost u načinima na koje studenti percipiraju koncept objekata u programiranju ukazuje na važnost raznovrsnih pristupa nastavi koji će zadovoljiti potrebe različitih učenika i olakšati njihovo razumijevanje ovog ključnog koncepta.


8. Kako doživljavate pojam objekta u kontekstu objektno-orijentiranog programiranja. Pokušajte navesti što više ključnih riječi i izraza kojima bi opisali objekt.

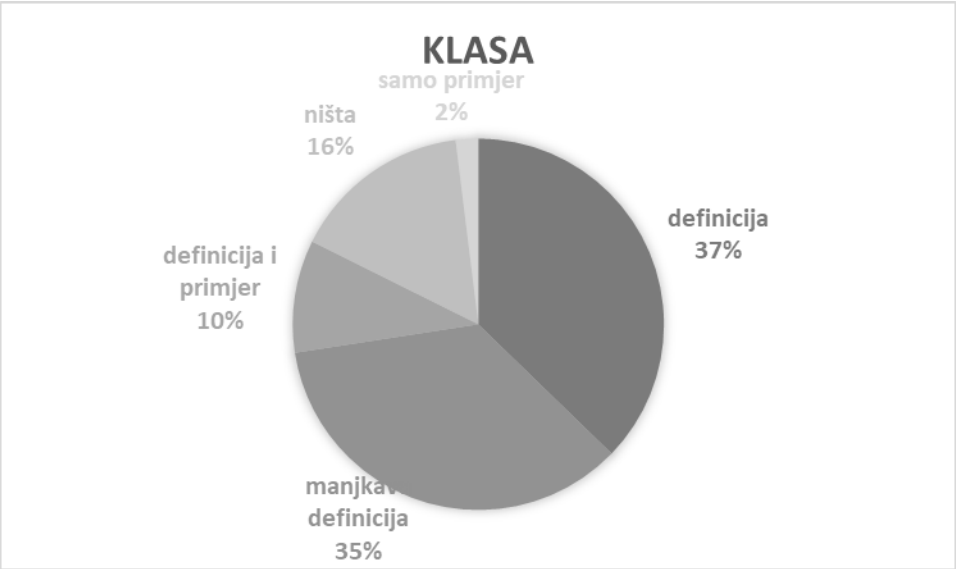
Objekt je instanca klase,
Životinja: klasa bi bila cijet ; sa njegova obilježja, a objekt primjerak cvijeta.

Slika 12 Primjer odgovora o klasi

Još neki od odgovora su „Objekt u sebi sadrži više podataka o tom objektu.“, „Objekt je instanca klase, konkretno nešto s čime se može raditi ili se može obrađivati i sadrži sva svojstva svoje pripadajuće klase.“, „Instanca klase primjerak korisnički definiranog tipa podatka.“, „Objekt instanca klase koji ima svoj identitet, ponašanje i stanje.“, „Objekt izvršava određene funkcija ili metode definirane unutar klase.“, „Klasa životinje, objekt, mačka pas.“, „Objekt je vrsta neke klase instanca cjelina na nekim temeljima klase.“, „Objekt je primjer klase a koje glase na primjer životinja. Onda objekt može biti neka konkretna životinja, na primjer pas sa svojim imenom.“, „Objekt je temelj objektno orijentiranog programiranja. Svaki objekt ima identitet, stanje i ponašanje.“, „Element kojim se upravlja.“.

3.5.9. Pojam klase

Najčešće što studenti koriste za opisivanje klase su definicija (19),a manjkavu definiciju koristi njih 18. Definiciju i primjer je upotrijebilo njih 5, samo primjer njih 8, a ništa nije napisala 1 osoba.

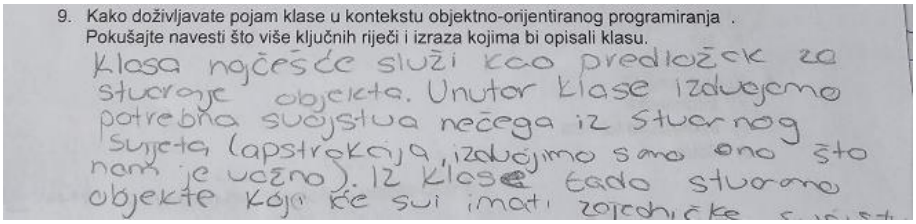


Slika 13 Prikaz izjava o klasi u postotcima

Analizom ključnih riječi koje su studenti koristili u izjavama o konceptu klase u programiranju, ocrtavaju se različiti stupnjevi razumijevanja i izazovi s kojima se suočavaju u tom procesu.

- Definicija: Za 37.3% studenata, naglasak je na jasnom razumijevanju definicije klase u programiranju. Ovi studenti prepoznaju važnost temeljnog definiranja koncepta kako bi mogli uspješno primijeniti klasu u svojem kodu.
- Manjkava definicija: 35.3% studenata izražava nedostatak potpunog razumijevanja definicije klase, što predstavlja izazov u njihovom učenju i primjeni ovog ključnog koncepta.
- Definicija i primjer: Za 9.8% studenata, kombinacija jasne definicije i primjera ključna je za njihovo razumijevanje koncepta klase. Ovi studenti preferiraju praktičan pristup uz teorijsko razumijevanje kako bi bolje internalizirali koncept.
- Ništa: 15.7% studenata izražava nedostatak bilo kakvog razumijevanja ili definicije klase u programiranju. Ovi studenti zahtijevaju dodatnu podršku i obrazovne resurse kako bi savladali ovaj koncept.
- Samo primjer: Manji postotak studenata (2.0%) ističe važnost primjera u razumijevanju klase, bez naglaska na teorijsku definiciju.

Raznolikost u načinima na koje studenti percipiraju koncept klase u programiranju ukazuje na potrebu za raznovrsnim pedagoškim pristupima koji će zadovoljiti različite potrebe učenika. Pružanje jasnih definicija, primjera i podrške može biti ključno za uspješno savladavanje ovog ključnog koncepta u programiranju.

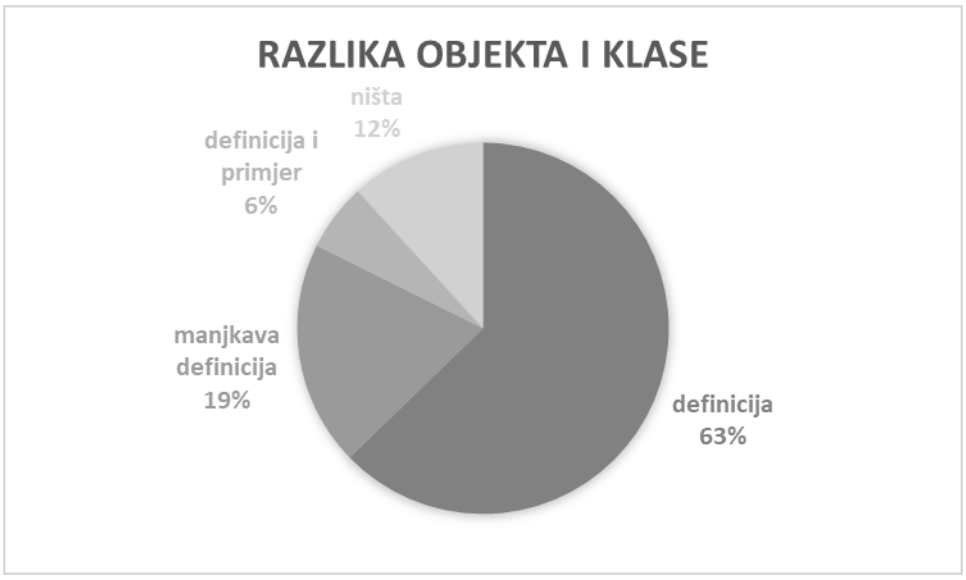

9. Kako doživljavate pojam klase u kontekstu objektno-orijentiranog programiranja .
Pokušajte navesti što više ključnih riječi i izraza kojima bi opisali klasu.
Klasa najčešće služi kao predložak za stvaranje objekta. Unutar klase izdvajamo potrebna svojstva nečega iz stvarnog svijeta (apstrakcija, izdvojimo samo ono što nam je važno). Iz klase tada stvaramo objekte koji će sui imati zajednička

Slika 14 Primjer odgovora o klasi

Još neki od odgovora su „Klasa je više objekata.“, „Klasa je skup osnovnih vrijednosti kao kostur na objekt pa se objekt nadograđuje svaki različit.“, „Klasa ima mogućnosti nasljeđivanja.“, „Skup elemenata kojima je potrebno upravljati, na primjer životinja, objekti, pas mačka, konstruktor svojstva, polje.“, „Klase su funkcije koje sadržaj, konstruktor i može se koristiti i više različitih podataka i ima opcija nasljeđivanja.“, „Klasa je skup specifikacija, tj. svojstava i funkcija koje opisuju objekte.“, „Korisnički definirani tip podatka koji služi kao forma za stvaranje više primjeraka slične ili iste vrste.“, „Klasa je više objekata u klasi definiramo neka svojstva koja su zajednička svim objektima.“, „Klasa najčešće služi kao predložak za stvaranje objekta. Unutar klase izdvajamo potrebna svojstva nečega iz stvarnog svijeta(apstrakcija, izdvojimo samo ono što nam je važno) iz klase tada stvaramo objekte koji će imati zajednička svojstva.“, „Klasa je korisnički definirani tip podatka. Instance klase su objekti.“, „Klasa je korisnički definirani tip podatka.“.

3.5.10. Razlika objekta i klase

Većina studenata (32) prepoznaje razliku između klase i objekta kao razliku između definicije (klasa) i instance (objekt). Drugi česti opisi sadrže manjkavu definiciju (10), definiciju s primjerom njih 3, ništa nije napisalo njih 6.

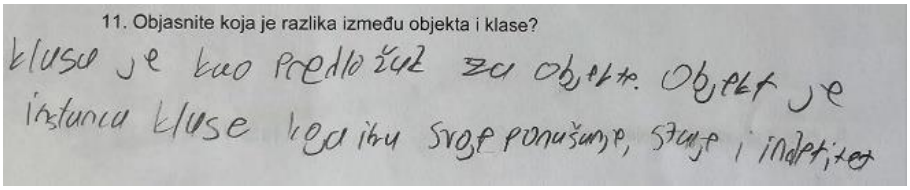


Slika 15 Prikaz izjava razlike klase i objekta u postotcima

Analizom ključnih riječi korištenih u izjavama studenata o razlici između objekta i klase u programiranju, ističu se različiti stupnjevi razumijevanja i izazovi s kojima se studenti suočavaju u razlikovanju ovih ključnih koncepata.

- Definicija: Za 62.7% studenata, naglasak je na jasnom razumijevanju definicije razlike između objekta i klase. Ovi studenti prepoznaju važnost temeljnog razlikovanja ovih pojmova kako bi uspješno primijenili ove koncepte u svojem programiranju.
- Manjkava definicija: 19.6% studenata izražava nedostatak potpunog razumijevanja razlike između objekta i klase, što predstavlja izazov u njihovom učenju i primjeni ovih koncepata.
- Definicija i primjer: Za 5.9% studenata, kombinacija jasne definicije i primjera ključna je za njihovo razumijevanje razlike između objekta i klase. Ovi studenti preferiraju praktičan pristup uz teorijsko razumijevanje kako bi bolje internalizirali koncept.
- Ništa: 11.8% studenata izjavljuje da nemaju nikakvo razumijevanje razlike između objekta i klase. Ovi studenti zahtijevaju dodatnu podršku i obrazovne resurse kako bi savladali ovu razliku i uspješno primijenili koncepte u praksi.

Raznolikost u načinima na koje studenti percipiraju razliku između objekta i klase u programiranju sugerira potrebu za raznovrsnim pedagoškim pristupima koji će zadovoljiti različite potrebe učenika. Pružanje jasnih definicija, primjera i podrške može biti ključno za uspješno razlikovanje i primjenu ovih ključnih koncepata u programiranju. Čest problem među početnicima programerima, koji također spominju Holland i suradnici, je razumijevanje razlike između klase i objekta. To je očigledno problem ako se prikazuju primjeri u kojima se koristi samo jedan primjerak svake klase. Da bi se to izbjeglo, dobra praksa je uvijek raditi s više primjeraka svake klase. [9]

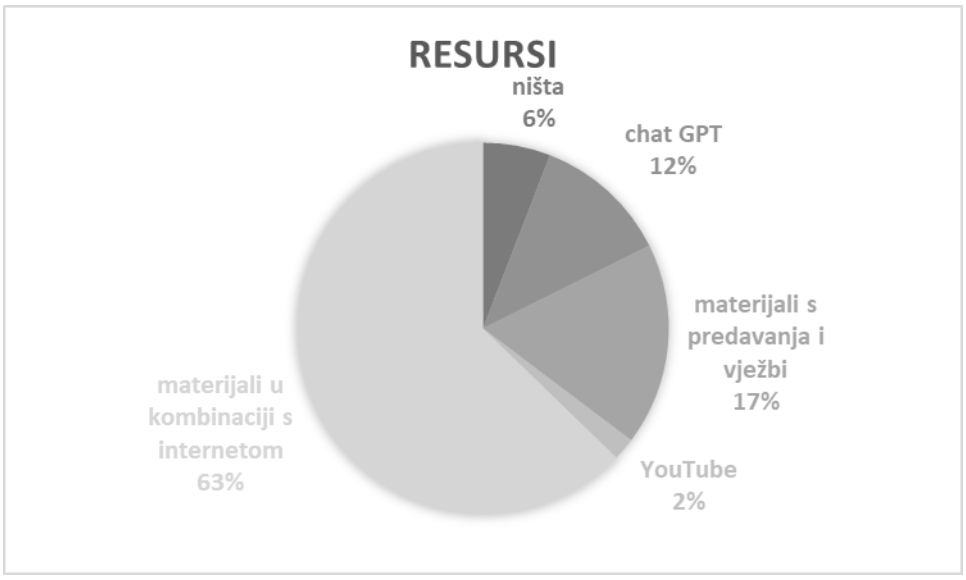


Slika 16 Primjer odgovora o razlici objekta i klase

Još neki od odgovora su „Objekt je instanca klase.“, „Klasa se može gledati kao nešto opće, dok je objekt konkretan primjer.“, „Klasa bi bila neka vrsta šablone koja će opisati karakteristike i ponašanje određenog tipa objekta. A objekt neka konkretna pojava te klase.“, „Objekt je instanca klase, a klasa ne može postojati bez objekta.“, „Klasa definira objekt, ponašanje i strukturu.“, „Objekt je instanca klase.“, „Objekt je instanca klase npr auto -> Tesla.“, „Objekt je primjerak izveden po pravilima klase.“, „Objekt je instanca klase. Klasa okuplja objekte s određenim (istim) svojstvima.“.

3.5.11. Resursi

Najčešći resursi koje studenti koriste za učenje OOP-a su materijali s predavanja i vježbi u kombinaciji s internetom (32), samo materijale s predavanja i vježbi njih 9. Manji broj studenata slanja se na ChatGPT (6) i YouTube 1 student, a ništa ne koristi 3 student.



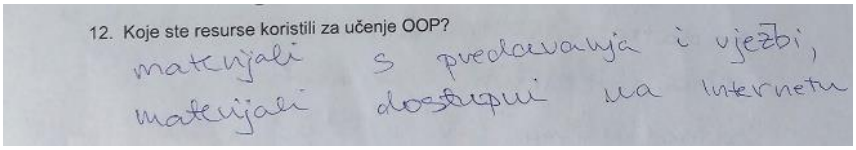
Slika 17 Prikaz izjava o resursima u postotcima

Analiza ključnih riječi korištenih u izjavama studenata o korištenju resursa u učenju programiranja otkriva različite načine na koje studenti pristupaju potrebnoj podršci i materijalima.

- Ništa: Za 5.9% studenata, nema korištenja vanjskih resursa u učenju programiranja. Ovi studenti vjerojatno se oslanjaju isključivo na resurse koje im pruža njihova obrazovna institucija ili samostalno istražuju.
- Chat GPT: 11.8% studenata koristi alate poput ChatGPT-a za pomoć i podršku u učenju programiranja. Ovo može uključivati postavljanje pitanja, traženje savjeta ili razmjenu ideja s umjetnom inteligencijom.
- Materijali s predavanja i vježbi: 17.6% studenata oslanja se na službene materijale s predavanja i vježbi koje pruža njihova obrazovna institucija. To uključuje udžbenike, prezentacije, zadatke i druge materijale koje su im dostupni od strane nastavnika.
- YouTube: Manji postotak studenata (2.0%) koristi YouTube kao resurs za učenje programiranja. To može uključivati tutorijale, predavanja, ili druge edukativne sadržaje dostupne na platformi.
- Materijali s predavanja i vježbi u kombinaciji sa internetom: Za većinu studenata (62.7%), kombinacija službenih materijala s predavanja i vježbi uz dodatne resurse s interneta predstavlja ključni izvor podrške u učenju programiranja. To ukazuje na

potrebu za raznovrsnim izvorima informacija i podrške kako bi se osiguralo sveobuhvatno razumijevanje gradiva.

Ova raznolikost u korištenju resursa od strane studenata naglašava važnost prilagodljivosti i raznolikosti u nastavnim metodama i materijalima kako bi se zadovoljile individualne potrebe učenika i podržalo njihovo uspješno napredovanje u učenju programiranja.



Slika 18 Primjer odgovora o korištenim resursima

Još neki od odgovora su „Materijali s predavanja i vježbi , YouTube i druge razne stranice po internetu.“, „Prezentacije i vježbe.“, „Materijali od profesora, YouTube, VSCode.“, „Bilješke s vježbi, prezentacije s predavanja, Internet, tutoriali na YouTube-u.“, „Prezentacije i rješavanje zadataka sa ChatGPT-om tj. on bi mi napisao kod i onda bi mi po kodu objašnjavao što se događa.“, „Prezentacije, tutoriali i skripte.“, „Prezentacije, Internet stranice i skripte.“, „ChatGPT, skripte kolegija, primjer zadataka, BroCode na YouTube-u.“.

3.6. Sažetak rezultata istraživanja

Fenomenografska analiza pruža vrijedne uvide u različite načine na koje studenti percipiraju i doživljavaju učenje OOP-a. Razumijevanjem tih percepcija, možemo bolje prilagoditi naše obrazovne prakse kako bi bile učinkovitije i pristupačnije svim studentima. Ova studija nastoji doprinijeti tom cilju kroz detaljno istraživanje i analizu studentskih percepcija i iskustava, s nadom da će rezultati pomoći u unapređenju nastave objektno orijentiranog programiranja i poboljšanju studentskog uspjeha u ovom ključnom području informatičkog obrazovanja.

Na temelju dobivenih informacija putem fenomenografske analize percepcije studenata o ciljevima i iskustvima u učenju objektno orijentiranog programiranja, možemo izvući nekoliko ključnih zaključaka:

- Raznolikost doživljaja učenja programiranja: Rezultati ukazuju na širok spektar percepcija studenata o procesu učenja programiranja. Dok većina studenata doživljava učenje programiranja kao pozitivno iskustvo, postoji značajan postotak studenata koji percipiraju izazove i stresne situacije tijekom učenja.
- Različiti ciljevi učenja programiranja: Ciljevi studenata u učenju programiranja variraju od osobnog razvoja i napretka u vještinama do stjecanja posla u IT industriji ili razvoja vlastitih projekata. Ovo odražava različite motivacije i svrhe koje studenti imaju u vezi s učenjem programiranja.
- Izazovi učenja programiranja: Studenti identificiraju različite izazove u procesu učenja programiranja, uključujući pisanje koda, veliku količinu gradiva u kratkom roku, nedostatak prakse, pronalaženje motivacije te proces debugiranja. Ovi izazovi zahtijevaju prilagođene strategije podrške kako bi se olakšao proces učenja i potaknulo uspješno napredovanje studenata.
- Raznolikost u percepciji ključnih koncepata OOP-a: Studenti različito percipiraju ključne koncepte objektno orijentiranog programiranja, poput objekata i klasa. Dok neki studenti uspješno definiraju ove koncepte, drugi pokazuju manjak razumijevanja ili koriste neadekvatne primjere. Ovo ukazuje na potrebu za dodatnim pojašnjenjem i podrškom u nastavi kako bi se osiguralo jasno razumijevanje temeljnih koncepata OOP-a.

Zaključak

Razumijevanje kako studenti doživljavaju učenje OOP-a ima višestruke koristi. Prvo, može pomoći nastavnicima da prilagode svoje metode poučavanja kako bi bolje odgovarale potrebama i očekivanjima studenata. Na primjer, studenti koji vide OOP kao alat za rješavanje stvarnih problema mogu više cijeniti praktične projekte i primjere iz stvarnog svijeta, dok oni koji su fokusirani na teorijske aspekte mogu imati koristi od detaljnijih objašnjenja i formalnih definicija. Percepcije studenata o ciljevima programiranja mogu utjecati na njihovu motivaciju i angažman. Studenti koji vide programiranje kao ključnu vještinu za svoju buduću karijeru mogu biti više motivirani da ulože dodatni trud i savladaju izazove, dok oni koji ga doživljavaju kao apstraktni ili irelevantni zadatak mogu biti manje angažirani i skloniji odustajanju.

Analiza studentskih iskustava može otkriti uobičajene prepreke i izazove s kojima se suočavaju tijekom učenja OOP-a. Ove informacije mogu biti korisne za razvoj dodatnih resursa i podrške koja bi mogla pomoći studentima da prevladaju ove prepreke i postignu bolje rezultate.

Prema rezultatima naglašena su pretežno pozitivna iskustva studenata s učenjem OOP-a. Ključne riječi koje najbolje opisuju ova iskustva su "zanimljivo" i "loša organizacija," dok se manji broj studenata suočava s osjećajem stresa, dosade i zahtjevnosti.

Većina studenata je doživjela proces učenja OOP-a kao zanimljiv i interaktivan, što je vrlo pozitivan pokazatelj angažmana i interesa. Međutim, značajan broj studenata istaknuo je lošu organizaciju kao problem, što ukazuje na potrebu za poboljšanjem strukture i pristupa predavanjima i vježbama. Manji broj studenata je naveo da im je učenje bilo stresno, dosadno ili prezahtjevno, što sugerira da postoje individualne razlike u percepciji zahtjevnosti ovog predmeta.

Većina studenata je usmjerena na napredak i karijeru, što je pokazatelj zdravog i produktivnog pristupa učenju. Pored toga, mnogi studenti žele isprogramirati vlastite ideje, dok su na kraju ciljevi kao što su polaganje kolegija i potencijalna zarada.

Analiza je pokazala da većina studenata može pružiti osnovne definicije klasa i objekata, ali rijetko potkrepljuju te definicije konkretnim primjerima. Ova praznina u razumijevanju

može otežati primjenu teorije u praktične zadatke, te je stoga ključno poraditi na integraciji teorijskih znanja s praktičnim primjenama. To se može postići kroz povećanje broja primjera u nastavi, dakle pružanje više praktičnih primjera koji ilustriraju definicije klasa i objekata. Također uvođenjem interaktivnih vježbi gdje studenti mogu odmah primijeniti naučeno. Pomoću redovite povratne informacije o studentskim radovima potencijalno bi se ispravili eventualni nesporazumi ili nedostatci u razumijevanju. Kako bi se studentima pomoglo da razlikuju sve različite aspekte fenomena, ova studija ističe važnost pružanja prilike studentima da prate cijeli programski zadatak, uključujući analizu problema, dizajn, implementaciju i testiranje programa. [9]

Većina studenata koristi kombinaciju službenih materijala i interneta za učenje, što ukazuje na fleksibilan pristup učenju i spremnost na korištenje različitih izvora informacija.

Na kraju, ovi nalazi ukazuju na potrebu kontinuiranog poboljšanja nastavnog procesa i prilagodbe metoda poučavanja kako bi se studenti bolje pripremili za praktične izazove i ispunili svoje ciljeve učenja.

U konačnici, fenomenografska analiza omogućila je identifikaciju različitih načina na koje studenti percipiraju ciljeve i iskustva u učenju objektno orijentiranog programiranja te ukazala na važnost prilagođavanja nastavnih metoda i strategija kako bi se potaknulo pozitivno shvaćanje učenja programiranja i podržao uspjeh studenata u ovom području. Istraživanje je zaista pružilo vrijedne uvide koji mogu doprinijeti poboljšanju obrazovnih praksi i strategija učenja.

Literatura

[1] R. B. Nagineni, A Research on Object-Oriented Programming and Its Concepts, Andhra Pradesh, India: International Journal of Advanced Trends in Computer Science and Engineering, 2021.

[2] J. Holvikivi, »Conditions for successful learning of programming skills,« u *IFIP TC 3 International Conference on Key Competencies in the Knowledge Society*, Brisbane, Australia, 2010..

[3] E. B. P. &. W. S. Janke, »Does outside-in teaching improve the learning of object-oriented programming,« 2015. .

[4] R. L. Timothy C. Lethbridgea, Object-Oriented Software Engineering: Practical Software Development using UML and Java, Maidenhead: McGraw-Hill Education, 2005..

[5] I. Sommerville, Software Engineering, University of Lancaster, United Kingdom , University of St Andrews, Scotland: Pearson, 2016.

[6] N. &. B.-A. M. Ragonis, A long-term investigation of the comprehension of OOP concepts, Rehovot: Computer Science Education, 2007.

[7] R. A. M. M. W. E. B. J. Y. P. J. C. K. A. H. Grady Booch, Object-oriented analysis and design with applications, Addison-Wesley Professional, 2008.

[8] R. a. M. B. Pressman, Software Engineering: A Practitioner’s Approach, McGraw-Hill. Boston, 2014.

[9] A. Eckerdal, Novice Students’ Learning of, Uppsala, Sweden: UPPSALA UNIVERSITY, 2006.

Tablica slika

Skraćenice

OOP	*Object oriented programming*	objektno orijentirano programiranje