Kriminalističko-policijska akademija, Beograd

Pisanje programa u Pythonu uz pomoć UML dijagrama

Slobodan Nedeljković
Mihailo Jovanović

Školska 2017/2018. godina

Sadržaj

Uvod

U ovom radu će biti opisana dva pristupa pisanja programa u kombinaciji UML dijagrama i Python programskog jezika. Oba pristupa se koriste u zavisnosti od potrebe i situacije sistema koji se projektuje, kao i u zavisnosti od znanja i iskustva projektanta. Pored ovoga biće dat i prikaz zašto je korisno koristiti Python u proeđenju sa drugim programskim jezicima i bitnos UML dijagrama.

Poređenje Python and Java

U Pythonu, nikad se ne deklariše ništa. Kada se naziv dodeli nekom objektu, a objekat može biti bilo kog tipa. Ako se naziv poveže sa nekim objektom, kasnije se može to isto ime iskoristiti za objekat drugog tipa. Na to se misli kada se kaže da kje Python dinamički kucan jezik.

U javi, sva imena promenjivih (zajedno sa njihovim tipovima) moraju biti eksplicitno deklarisini. Pokušaj da se dodeli pogrešan tip objekta promenjivoj određenog tipa prikazaće grešku. Na to se misli kada se kaže da je Java statički pisan jezik.

Objekti kao što su *Vector* i *ArrayList* koji se koriste kao kontejneri skladište objekte generičkog tipa *Object*, ali ne mogu da sadrže primitivne tipove kao što je *int*. Da bi sačuvao *int* u *Vector*, program mora konvertovati *int* u *Integer*. Kada se dohvati objekat iz kontejnera on se mora kastovati u željeni objekat jer kontejner ne pamti šta je sačuvao.

U Pythonu liste i rečnici (*list, dictionaries*) i drugi kontejneri mogu da skladište različith tipova uključujući i brojeve i liste. Kada se dohvati objekat iz kontejnera, on pamti koji je tip i nije potrebno kastovati ga.

U Javi svaka top-level javna klasa mora biti definisana u zasebnom fajlu. Ako je aplikacija sastojena iz 15 takvih klasa mora postojati i 15 fajla. Dok kad je reč o Pythonu više klasa mogu biti čuvani u jendom fajlu. Ako aplikacija ima 15 klasa, celu aplikaciju je moće smestiti u jedan fajl, iako je preporučljivo da se razdvoji u 4-5 fajlova.

Java je znatno slabija kad je reč o radu sa stringovima. Poboljsali su kvalitet uvošenjem dodatne split metode String klasi u Javi 1.4, ali svakako je u odnosu na Python to znatno uočljivo.

C i Python

C ima mogućnost upravljanja memorijom, Python nema. Ovo znači da postoje stvari koje C može da odradi a Python ne može. C radi brže nego što će Python ikad moći. U C-u je moguće pisati operativne sisteme i mnogo programe koji sa komponentama računara komuniciraju na nižem nivou, dok to u Pythonu nije moguće.

A to isto znači da postoji veliki broj grešaka i bagova koji se lako pišu a tešsko primećuju u C-u koje je nemoguće napisati u Pythonu. U Pythonu će dobro napisan kod uvek biti čitljiv i razumljiv u C-u će uglavnom biti komlikovan i težak za razumeti.

Zašto učiti Python programski jezik?

Python je savršen jezik za početnike u programiranju, iako postoji dugi niz godina idalje mu raste popularnost. Neke od najbitnijih karakteristika Pythona su:

- Lako čitljiv i lako se uči
- Veoma produktivan, kako za male tako i za velike projekte

- Velike biblioteke za razne stvari

Za šta se Python koristi?

Kao programski jezik opšte namene, Python se može koristiti za više stvari. Python se može koristiti za male ili velike projekte, onlajn ili oflajn. Najbolje opcije za korišćenje Python-a su razvoj veb aplikacija, jednostavne skripte i analiza podataka. Neke od mogućih primena za Pythona su:

Razvoj na veb-u:

Python se može koristiti za razvoj veb aplikacija na raznim nivoima složenosti. Postoje mnogi odlični frejmvorci koji omogućavaju jednostavan i kvalitetan razvoj veb aplikacija. Pyramid, Django, Flask su samo neki od njih.

Analiza podataka:

Python je prvi izbor za mnoge "data scientists". Python je postao sve popularniji u ovom polju zbog svojih odličnih biblioteka kao što su NumPy i Pandas, kao i biblioteke za vizualizaciju podataka Matplotlib i Seaborn.

Mašinsko učenje:

Postoje mnoge odlične biblioteke koje implementiraju alogritme za mašinsko učenje kao što su Sckit-Learn, NLTK i TensorFlow.

Računarski vid:

U Pythonu je pomoću OpenCv biblioteke omogućen je implementiranje algoritma za prepoznavanje lica, boja, slova i slično.

Intern stvari sa Raspberry Pi:

Raspberry Pi je vemo mal ii pristupačan kompjuter koji je razvijen u edukacione svrhe, a njegova popularnost u krugovima ljudi koji se bave hobijem uradi sam a vezano za računare i automatizaciju je u enormnom porastu. Razni roboti i automatizacije u kući su pravljene pomoću ovog rešenja. A programiranje ovog mozga raznih robota se radi u Pythonu.

Razvoj igrica:

Razvoj igrica sa modulo Pygame omogućava produkciju raznih igrica koji mogu da rade na različitim platformama.

Veb scraping:

Za potrebe skidanja podataka sa sajtova koji nemaju API za te podatke često se koristi Python.

Pisanje skripta:

Kad se ručno na računaru radi, nešto što se često ponavlja, kao što je pisanje mejlova, nije teško automatizovati ovakve procese samo sa osnovnim poznavanjem jezika.

Automatizacija pretraživača:

Neke od jednostavnih, a korisnik stvari kao što su otvaranje pretraživača i objavljivanje statusa na društvenim mrežama se mogu jednostavno postići pomoću Selenium-a i Pythona.

GUI razvoj:

Razvoj GUI aplikacija, tačnije desktop aplikacija sa modulima kao što su Tkinter ili PyQt koji je čak zasnovan na drag-and-drop sistemu.

Pisanje prototipa:

Python ima biblioteke za skoro bilo šta. Dobar je za brzo pravljenje prototipa, odnosno primera ili testova sa slabijim preformansama.

Python se može koristiti za razne projekte, ali mogu i drugi jezici, ono što njega izdvaja pored ogromnog broja biblioteka, brzine pisanja, čitljivosti, lakog učenja, dobre dokumentovanosti je i dobra i kvalitetna zajednica "community" koja daje dobru podršku prilikom pisanja bilo kog projekta kako za početnike tako i za iskusne programere.

Python can be used for any programming task, from GUI programming to web programming with everything else in between. It's quite efficient, as much of its activity is done at the C level. Python is just a layer on top of C. There are libraries for everything you can think of: game programming and openGL, GUI interfaces, web frameworks, semantic web, scientific computing...

Zašto koristiti UML

Unifikacija terminologije i standardizacija su omogućila veliko olakšanje u komunikaciji između svih članova koji sarađuju na nekom projektu. UML omogućava razmenu modela između različitih departmana u jednoj kompanij. Čak i više od toga, olakšava transfer projekta između projektnih timova. Predstavlja moćan alat za modelovanje, pomoću koga je moguću razvoj jednostavnih modela ili kompleksnih sistema. Pa tako se može krenuti od jednostavnih modela koji se kasnije po potrebi mogu dalje razvijajti u modele kompleksnih sistema. Zasnovan je na široko korišćenim i dokazanim pristupima. UML je razvijen da reši probleme sa kojima se oni koji projektuju i realizuju sistem i softver susreću.

Kada pisati UML dijagrame?

UML dijagrami se pišu tokom celog procesa pisanja programa. U fazi pripreme pisanja, samog pisanja i kada se napiše softver UML dijagrami se konstruišu kao standard u komunikaciji o funkcijama softvera, između programera, dizajnera, menadžera, marketinga i svih ostalih uključenih u razvoj aplikacije. U svakoj od navedene tri faze zivotnog ciklusa softvera konstruisanje UML dijagrama se radi iz drugačijih razloga.

Pre pisanja softvera, UML dijagrami se konstruišu da bi moglo da se govori o bibliotekama, potrebama hardvera i softvera koje je potrebno ustanoviti što ranije da bi se na vreme počelo sa izradom softvera.

Tokom samog razvoja softvera, dijagrami se konstruišu da bi se govorilo o tome kako komponente na nižim nivoima komuniciraju između sebe. Kako se kroz tok vremena se menja stanje aplikacije kao i sami resursi, podaci i baza koja se koriste treba kroz ove dijagrame vizualizovati. Ovakvi dijagrami se uglavnom prave da bi tim koji radi na razvoju aplikacije mogao razume na koji način treba da prilagode svoj deo sistema da bi sve moglo da funkcioniše kako je zamišljeno. Da članovi tima uoče nepravilnosti i na vreme isprave greške u sistemu i da diskutuju o kompleksnim problemima.

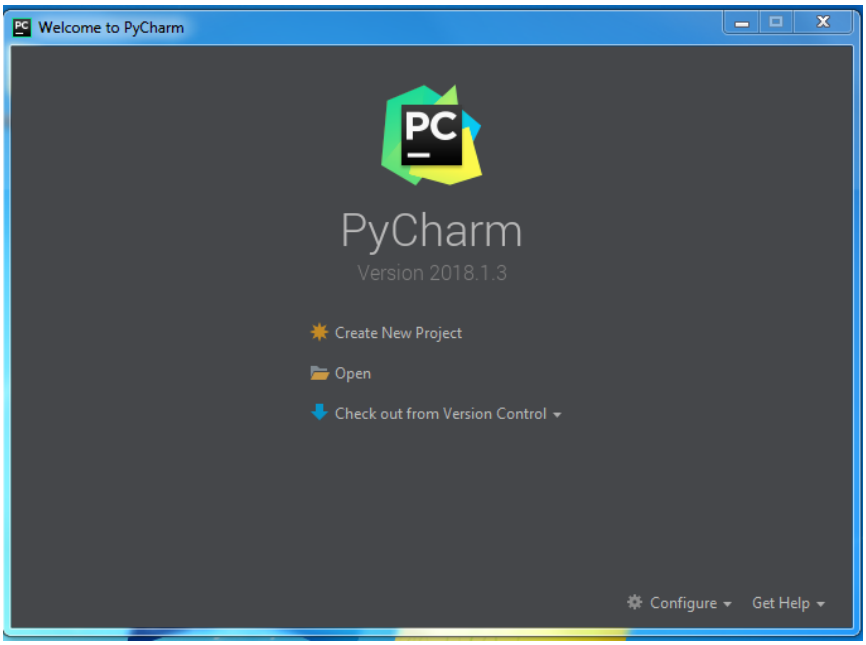
Nakon završetka pisanja softvera za razvoj samog softvera su najmanje bitni, jer je softver već napisan, ali se koriste kada je reč o daljem razvoju softvera i prilikom pravljenja dokumentacije i o pisanju uputstva za korišćenje softvera. Pored toga, UML predstavlja standard, pa tako da bi se podigo i kvalitet same aplikacije treba imati kvalitetnu dokumentaciju koja sadrži ovakve dijagrame.

Dijagrami pravljeni tokom razvoja same aplikacije će uvek biti najznačajniji i najvredniji jer će se tu uočiti greške i mane u sistemu, potencijalna unapređenja i pomoću ovakvih dijagrama je moguće optimizovati rad samog sistema.

U mnogim programskim jezicima, a naručito jezicima viseg nivoa kao što je Java, u potpunosti je moguće da celokupni objektno orjentisani dizajn se uradi unapred pomoću UML dijagrama. Klasni dijagrami su omogućili postizanje ovakvog pristupa.

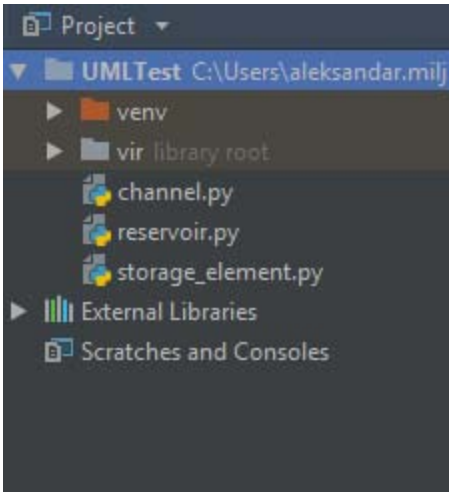
Mogućnost generisanja UML dijagrama iz koda

U ovom odeljku ce biti opisan način generisanja UML klasnog dijagrama u okviru PyCharm okruženja iz već napisanog Python koda. Ovo predstavlja pristup gde programer, uglavnom iskusniji u pisanju programa pristupa pisanju programa bez unapred vizualizovanog koda, a onda nakon što je program napisan, da bi bolje dokumentovao ono što je napisao i da bi na adekvatan način mogao da prezentuje sam dizajn softvera generiše UML klasni dijagram. Precedura opisana je moguće odraditi u PyCharm Professional verziji. Odabrana je ova procedura zato što se generisanje koda vrši automatski i sve promene direktno u kodu utiču automatski na dijagram, a PyCharm predstavlja jedno od vodećih okruženja za pisanje Pythona.



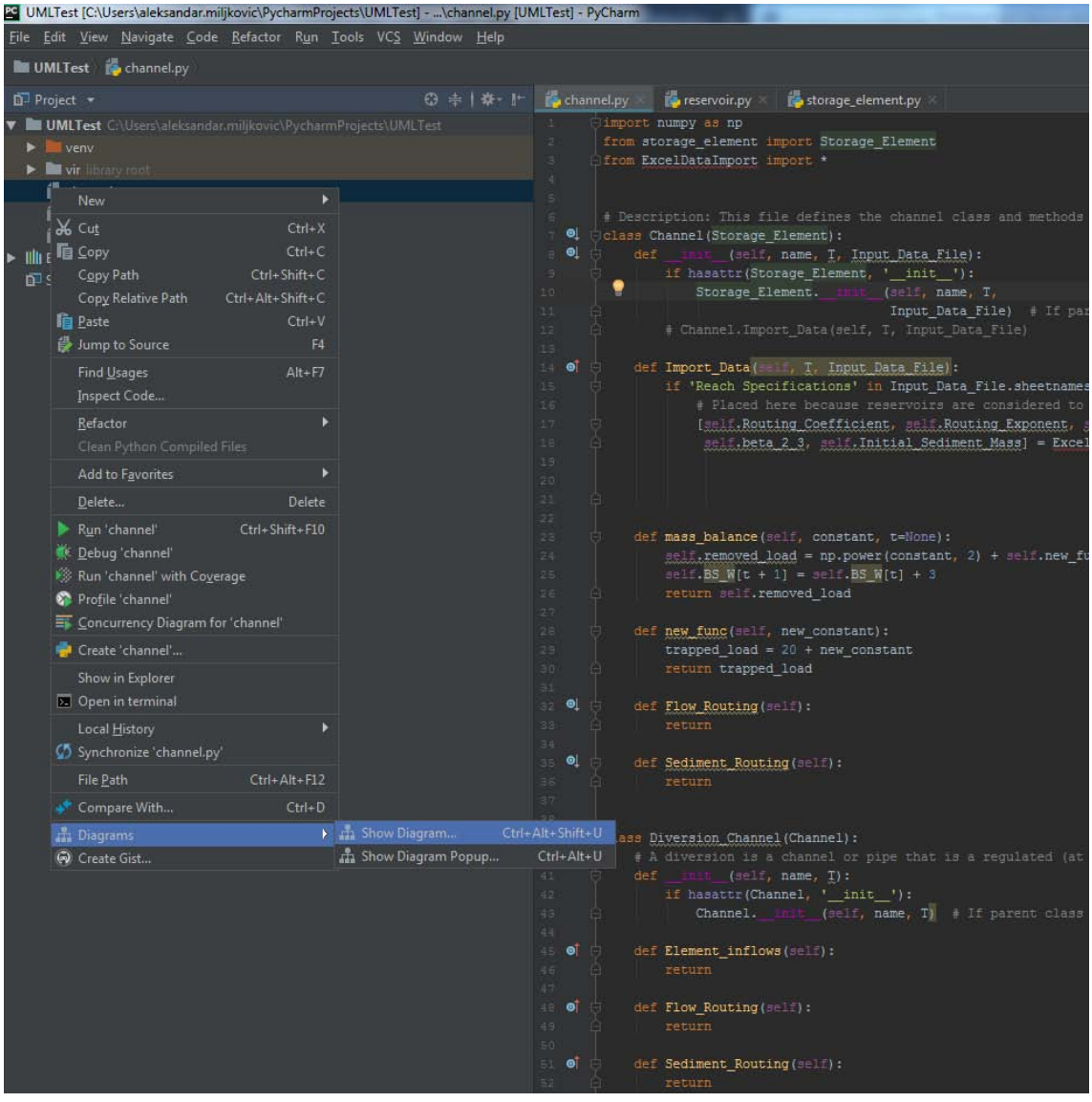
Slika 1 – Početni ekran prilikom pokretanja PyCharm alata

Prvi korak je kreiranje novog PyCharm projekta. Ovo se postiže pokretanjem aplikacije i klikom na “Create New Project” dugme. Ili u okviru startovanog projekta File -> New Project. Takođe moguće je i otvoriti postojući projekta i odraditi generisanje goda nad njime.



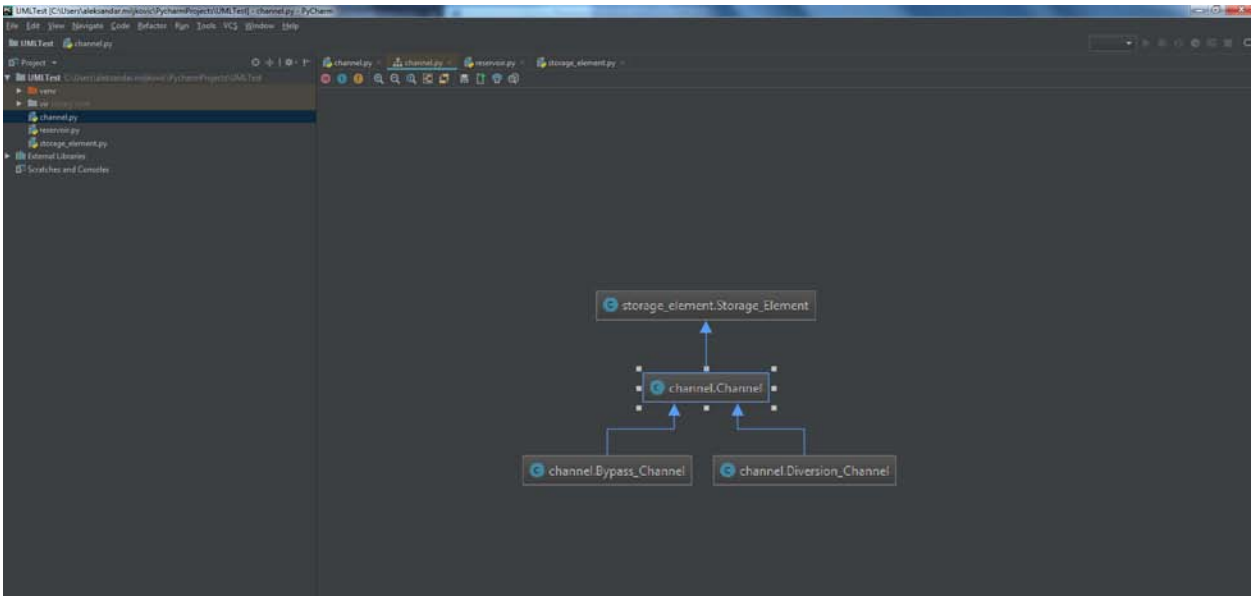
Slika 2 – Kreiran projekat sa fajlovima

Kada programer napiše svoj program, odnosno napise klase koje želi da koristi u svom program.



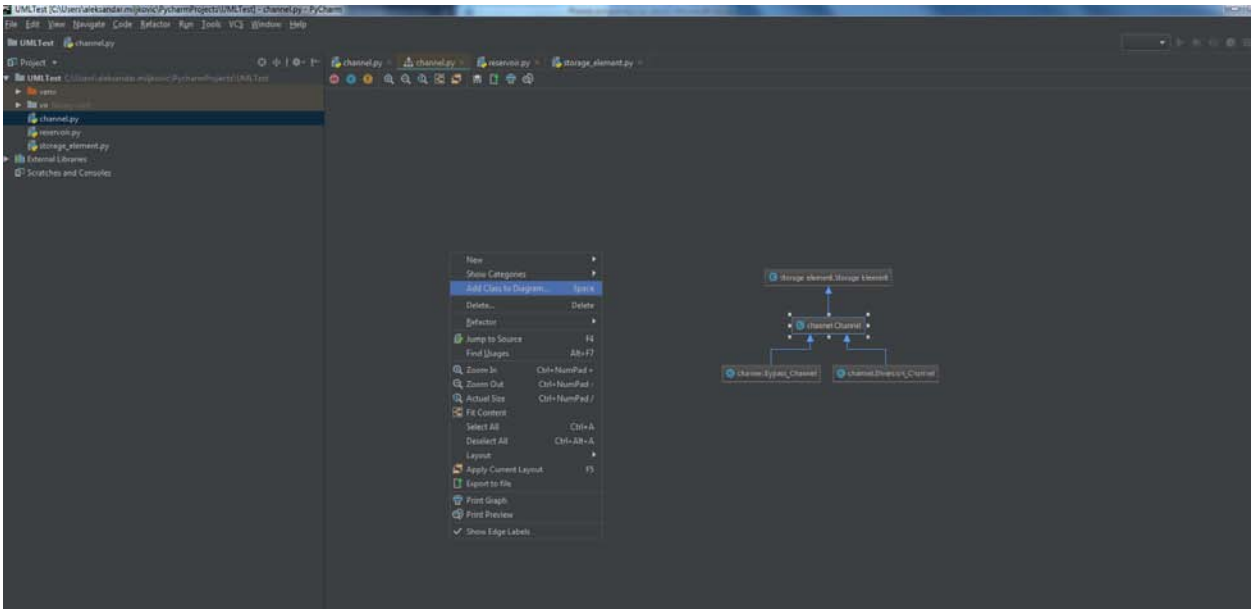
Slika 3 – Omogućavanje prikaza dijagrama

Desnim klikom na fajl koji sadrži klasu nudi se mogućnost prikaza dijagrama za datu klasu.



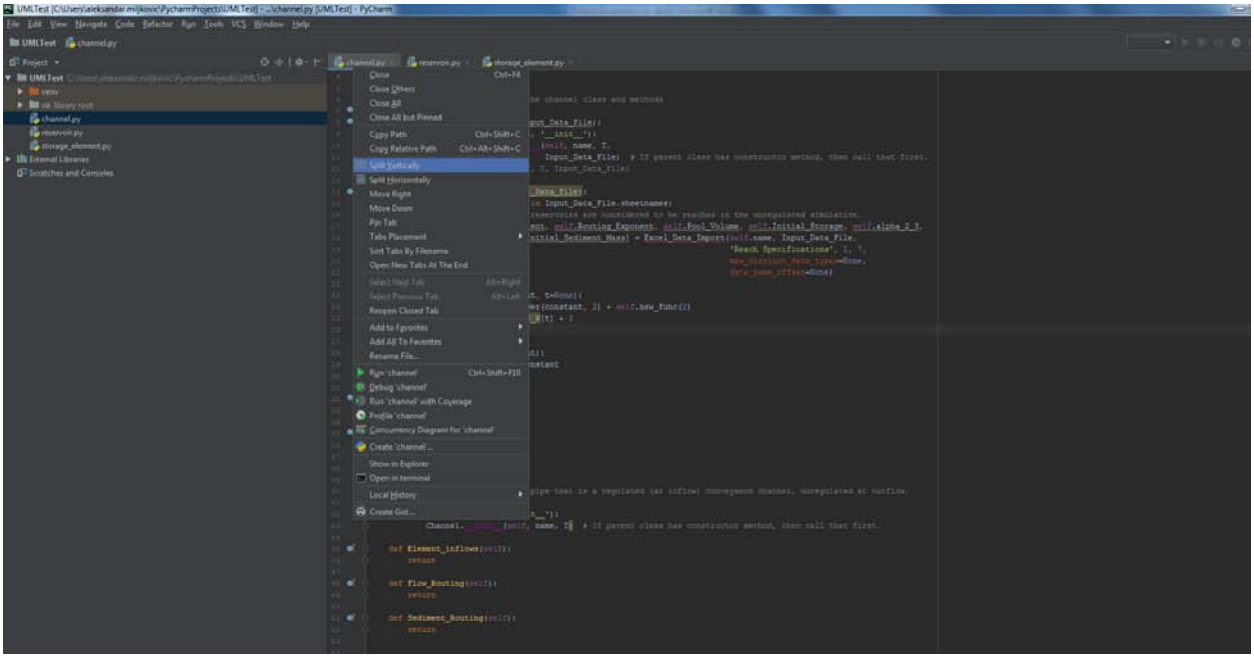
Slika 4 – Prikaz dijagrama

Pritiskom razmaka na tastaturi illi desnim klikom na pozadinu i klikom na Add Class to Diagram nam se omogućava da unesemo naziv klase koja se želi dodati u dijagram pored već odbrane.



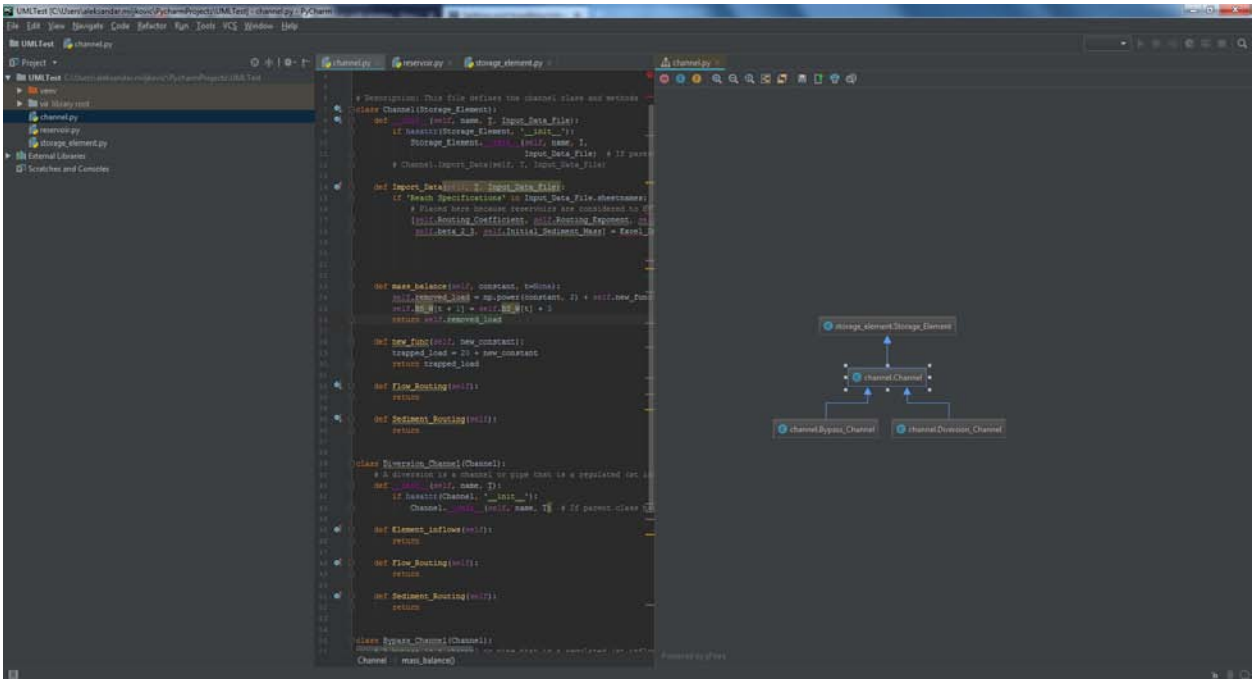
Slika 5 – Dodavanje klasa u dijagram

Ako želimo da direktno pratimo promene u našem kodu pomoću dijagrama možemo da pomoću Split screen funkcije podelimo ekran na dva dela i tako radimo u okruženju. Desnim klikom na tab koji želimo da podelimo i odaberemo split vertically.



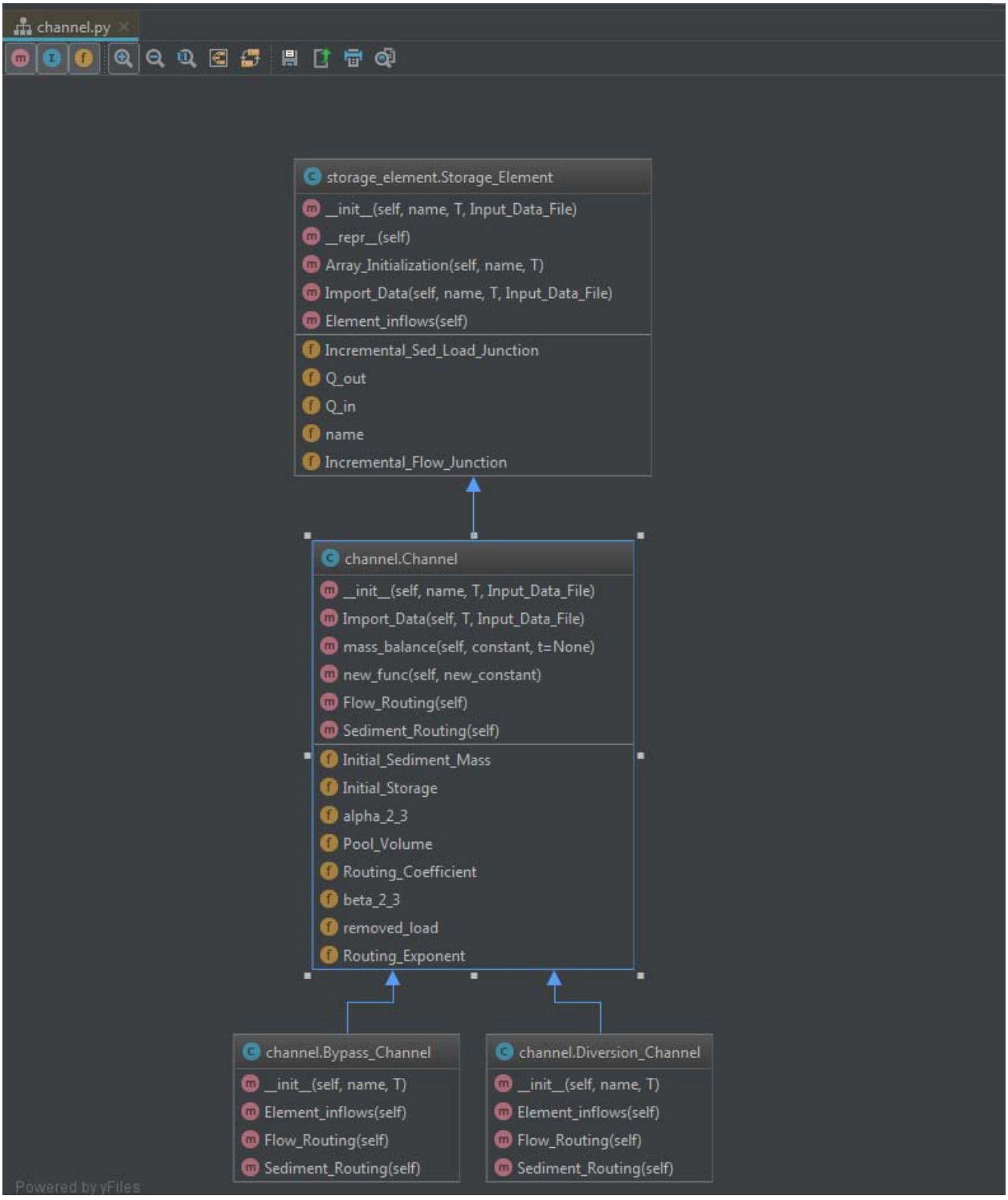
Slika 6 – Postupak podela ekrana radi preglednije analize

Na ovaj način vizualno je moguće ispratiti promene i time uočiti nepravilnosti ili čak uvideti optimizacije u realnom vremenu.



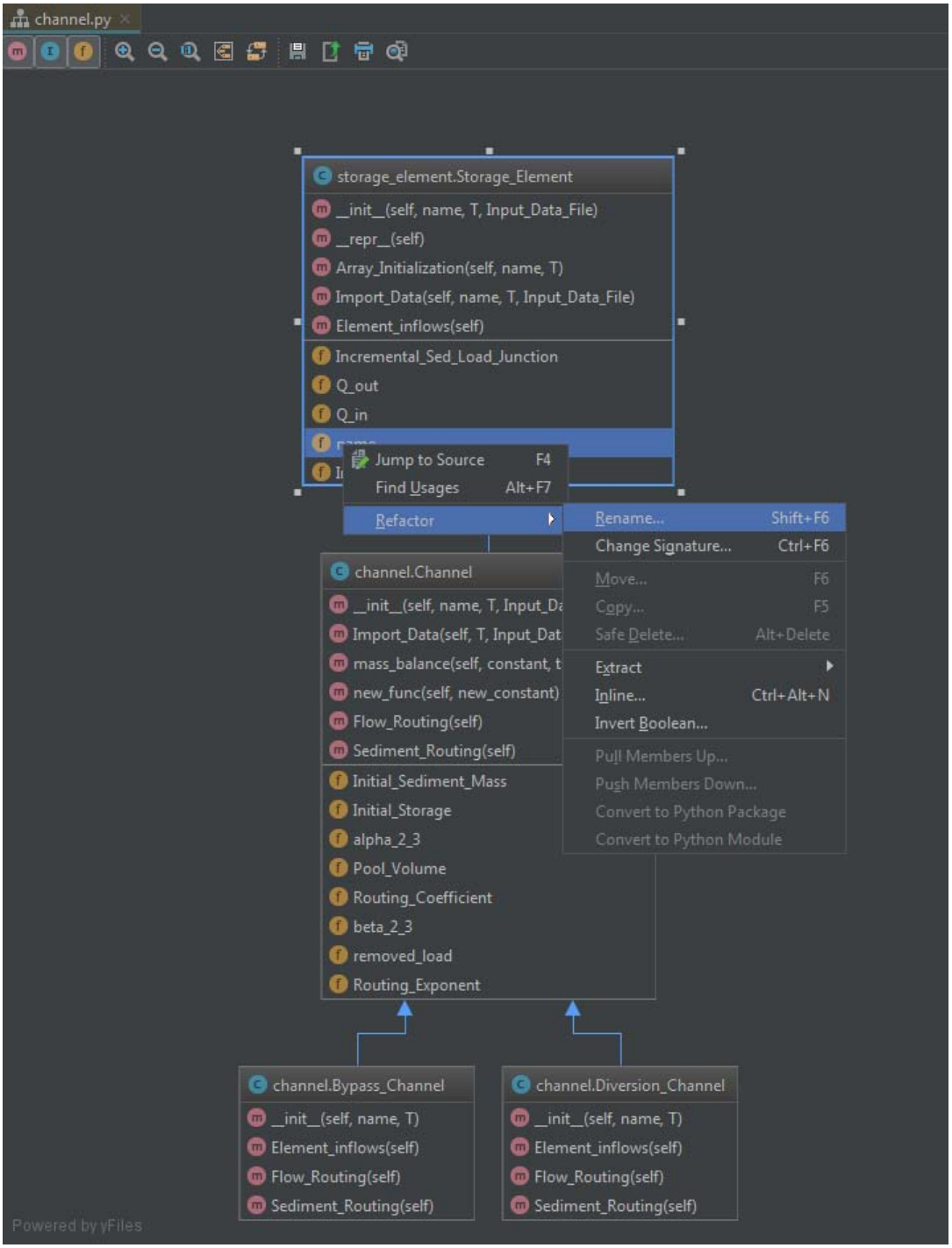
Slika 7 – Podeljen ekran

Dodatni prikaz metoda, unutrašnjih klasa i polja uključujemo klikom na jednu od tri ikonica u gornjem levom uglu (m, I, f).



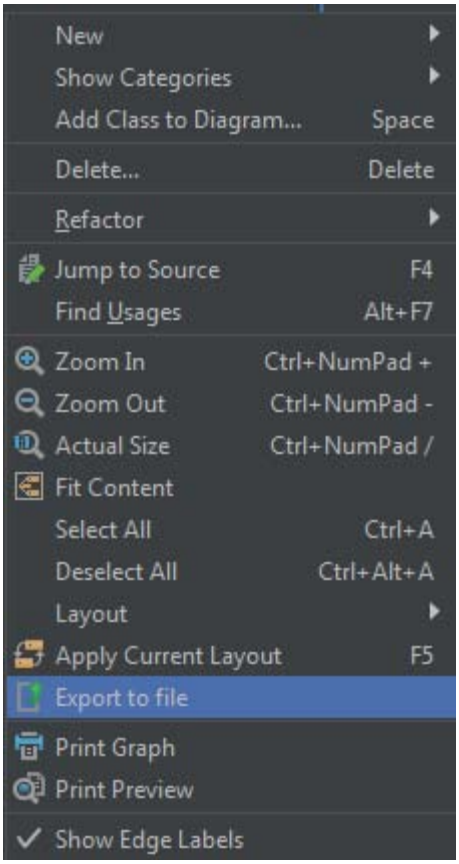
Slika 8 – Prikaz klasnog dijagrama sa svim detaljima

Dijagrami koji se prikazuju se ne koriste samo kao vizualno rešenje, već nude mogućnost nekih jednostavnijih izmena, kao što su promena naziva metoda ili promenjivih. Ovo postižemo desnim klikom na parameter koji želimo da izmenimo pa odaberemo refactor polje i iz menija odaberemo rename za pormenu imena. Na ovaj način na brzinu, vizaulno možemo da manipulišemo kodom.



Slika 9 – Izmena naziva atributa iz dijagrama

Konačno kad je kod gotov, i dijagram uređen na način kako nam to najviše odgovara, klikom na dijagram i odabirom Export to file polja omogućava se izvoz samog dijagrama koje se može koristiti u dokumentaciji ili u daljem razvoju aplikacije.



Slika 10 – Izvoz dijagrama u fajl

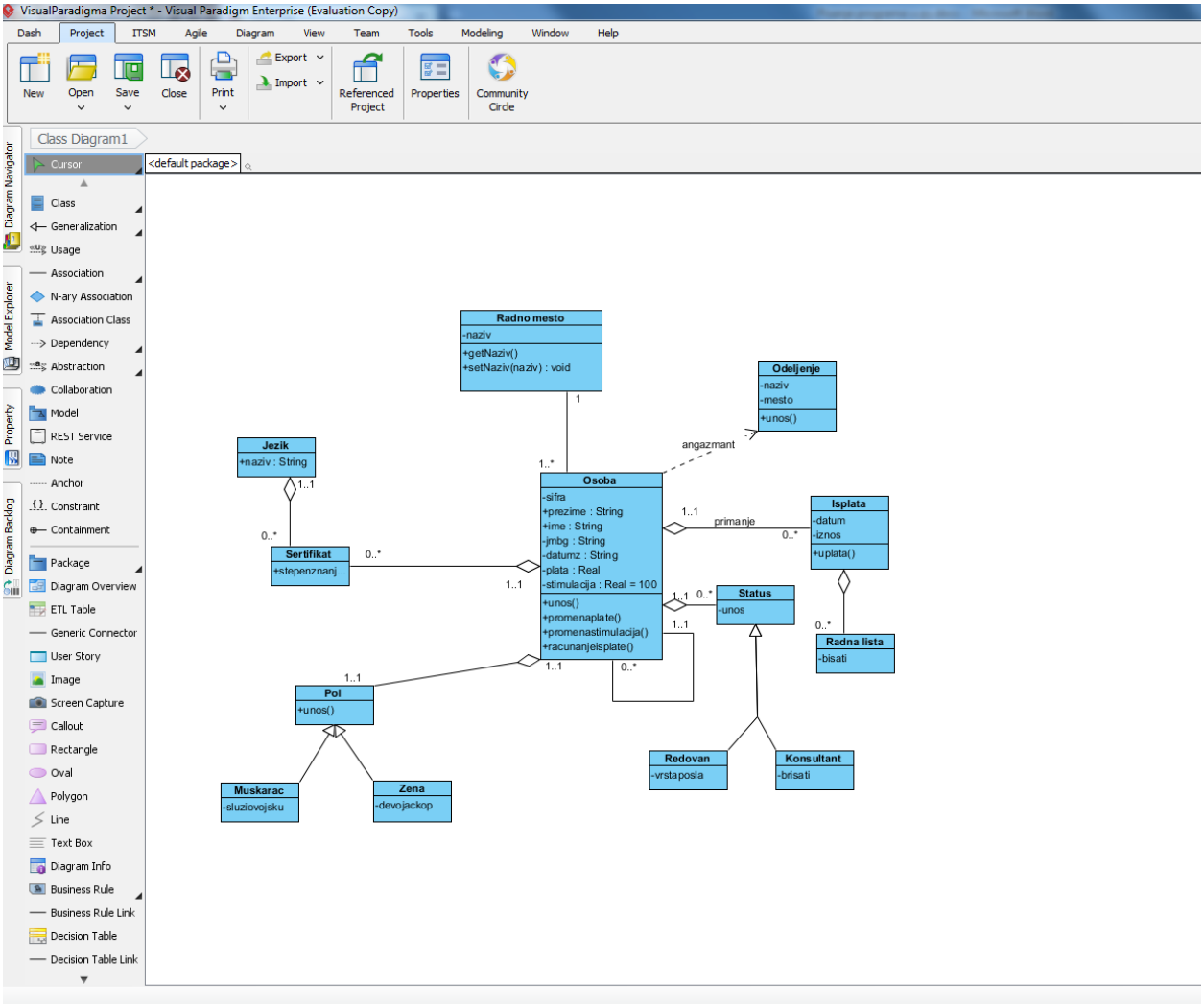
Ovim zaokuružujemo metodu generisanja i menajanje UML dijagrama iz Python koda. Ovakav jedan pristup je karaterističan za programere, dok bi pristup u kome bi projektant sistema, ili menadžer pristupio prvo projektovanju kroz UML a onda bi iz UML dijagrama generisao kod. Kombinacija Pythona i UML dijagrama na ovaj način omogućava brzo i jednostavno projektovanja kroz prototip od koga bi kasnije jedan sistem bio implementiran u nekom drugom okruženju.

Kako generisati Python kod iz UML dijagrama?

Automatska generacija je process proizvodnje izvornog koda iz klasnog modela. Dizajneri ili projektanti mogu da naprave na višem ili osnovnom nivou klasni model i onda ga proslede programeru da radi niže, odnosno konkretnije sistemske poslove ili modelovanje aplikacija na osnovu prosleđenih klasa dobijenih iz klasnih dijagrama. Ovakav pristup omogućava da pisanje softvera bude brže i jeftinije.

Jedan od alata koji omogućava ovakav pristup je VisualParadigma. On nudi izradu raznih dijagrama i izvrstan je alata za izradu dijagrama za kompleksne sisteme. Alat koji se koristi od strane projektanta.

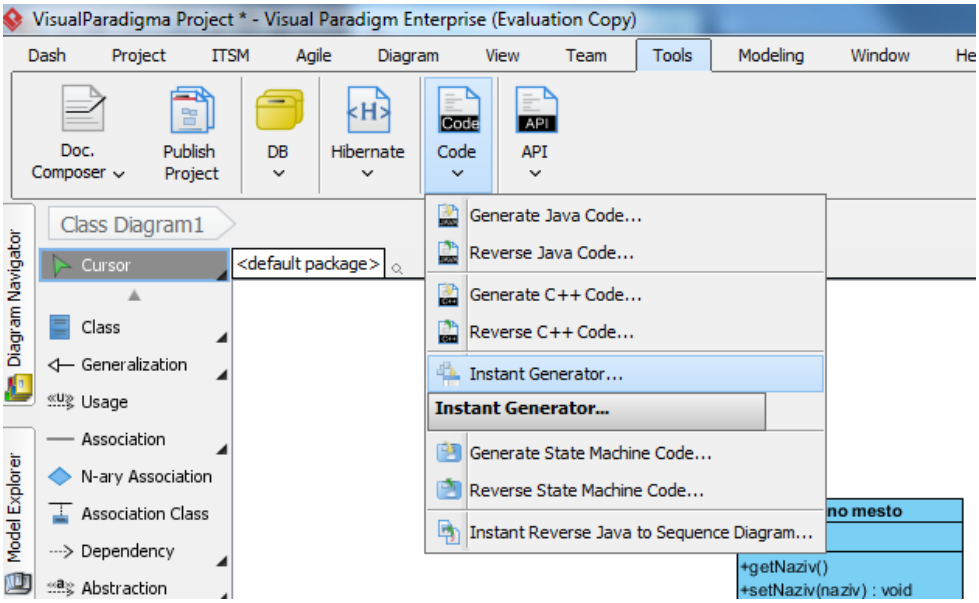
Između ostalih tipova dijagrama moguća je izrada i UML klasnih dijagrama. Projektant pristupa projektovanju nekog sistema kroz izradu jednog klasnog dijagrama.



Slika 11 – Konstruisan klasni dijagram

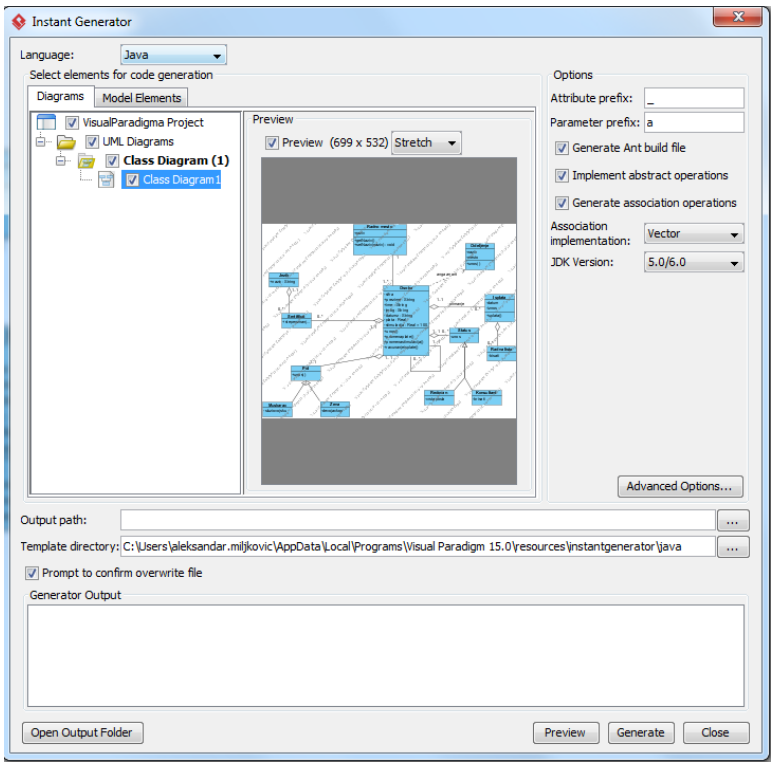
Nakon dizajniranja klasnog dijagrama, u kome može da opiše attribute i metode, kao i veze koje treba implementirati, da bi ubrzao taj proces, projektatnt može da izgeneriše kod koji programer može da iskoristi i time skrati vreme potrebno da se sistem izradi.

Generisanje koda se radi tako što se koristi Instant Generator alt u okviru softvera. Da bi se pristupilo ovom alatu ide se na karticu Tools i odatle se odabira podmeni Code i bira se alat Instant Generator.



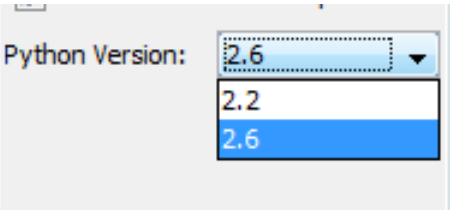
Slika 12 – Pokretanje Instant Generator dodatka

Ovde se otvara prozor u kojem se odabira u koji programski jezik je potrebno generisati kod, koje dijagram koristiti, putanju, kao i druge parametre relevantne za generisanje koda.



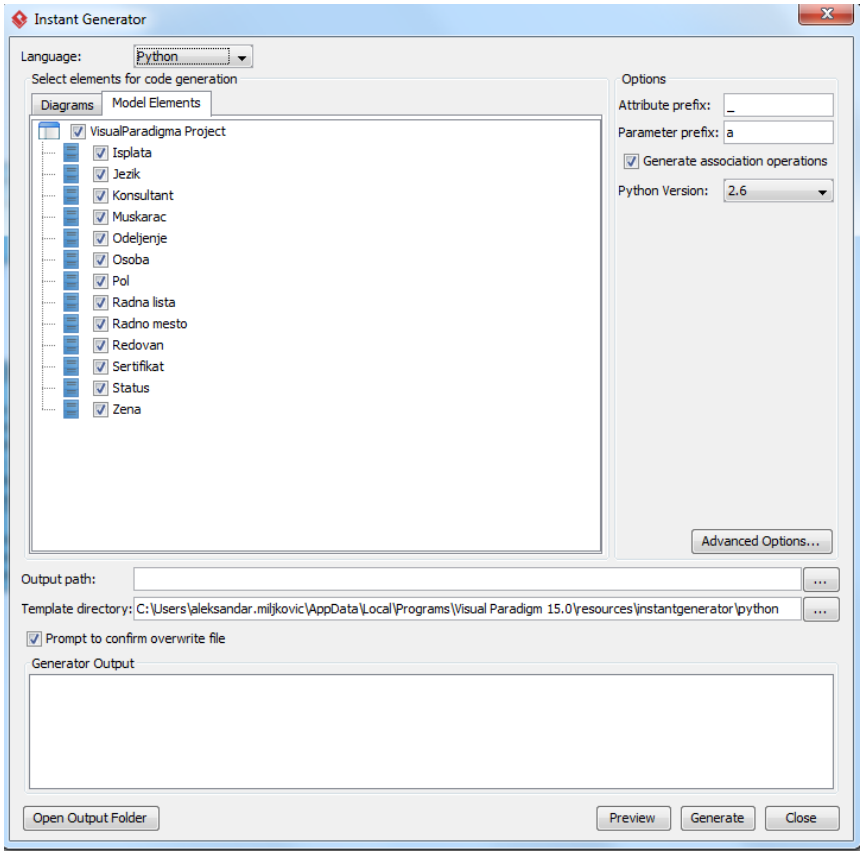
Slika 13 - Instant Generator

Prilikom odabira koda bitno je uočiti da Python koji se unosi je moguće izvesti samo u verziju koda 2.2 ili 2.6.

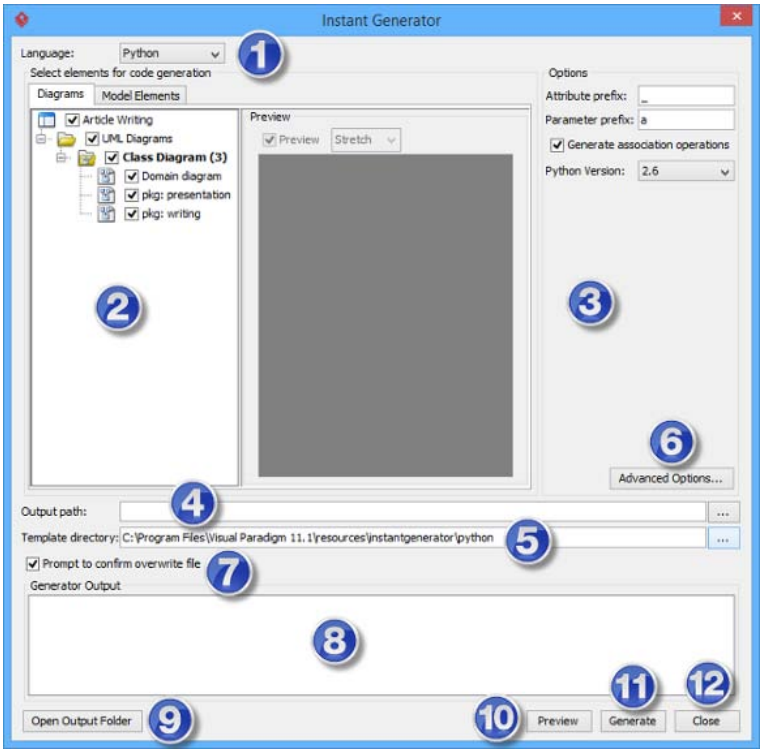


Slika 14 – Odabir verzije Python koda

Korisnik unosi podešavanja koja želi u kartici model elements odabira koje klase u okviru dijagrama želi da generiše.



Slika 15 - Odabir klasa koje korisnik želi da generiše iz klasnog dijagrama

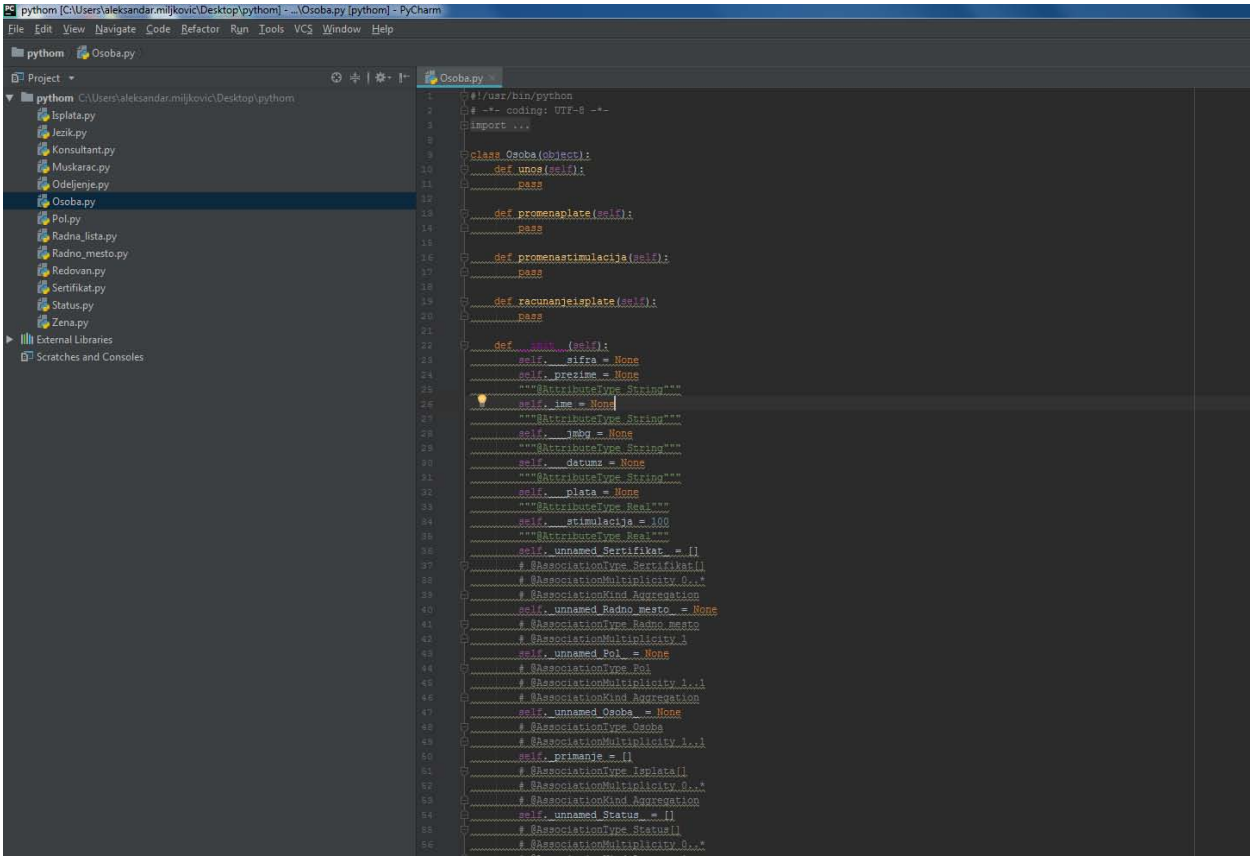


Slika 16 – Pregled podešavanja prilikom genersianja koda iz dijagrama

Tabela 1 - Tabela opisa

Broj	Naziv	Opis
1	Jezik	Programski jezik u kome se generiše kod.
2	Drvoj modela	Lista paketa i klasa koji mogu da se izaberu i izgenerišu. Moraju se odabrati klase da bi se generisao kod.
3	Generalna podešavanja	Neke od najčešće korišćenih podešavanja su ovde. Takođe se mogu konfigurisati i u naprednim podešavanjima.
4	Izvozna putanja	Folder gde će generisan kod biti sačuvan.
5	Odabir templejta	Temlejti omogućavaju kustomizaciju generisanog koda. Dodavanje dodatnih hedera, komentara ili drugih opcija.
6	Napredna podešavanja	Napredna ili dodatna podešavanja se otvaraju klikom na ovo polje.
7	Pitaj da li da prepiše postojući fajl	Ako već postoji kod fajl na lokaciji a ova opcija je odabrana korisnik će biti pitan pre prepisivanja postojućeg fajla, ako ova opcija nije odabrana, postojiući fajl će automatski biti prepisan.
8	Prikaz greški	Any warning, error or progress about generation will be printed here.
9	Otvori izvozni folder	Ovde korisnik bira folder u koji želi da sačuva generisan kod.
10	Prikaz koda	Prikaz koda koji će biti generisan, samo prikaz a ne i konkretno generisanje.
11	Generisanje	Pokreće se generisanje.
12	Izlaz	Zatvara se prozor za generisanje.

Kada se kod izgeneriše programer može početi sa dodatnom doradom koda. Definisanjeme koda samih metoda u okviru klasa.



Slika 17 - Generisan python kod iz dijagrama

Zaključak

Python kao programski jezik nudi pokrivenost raznih platform i tehnologija, a pored toga nudi jednostavnost pisanja, preglednost koga i brzinu učenja. U kombinaciji sa tehnologijama visualnog programiranja koje nudi UML dijagrami, put koji se porlazi od projektovanja sistema do krajnjeg softvera koji radi u ovom sistemu se zantno skarćuje. Pored efikasnijeg pisanja softvera, dokumentovanje kroz UML dijagrame, koji su danas standard, jako je bitna stavka u svakom sistemu, a korz ove primere smo uočili da su ovakvi pristupi idealni za tu vrstu dokumentovanja.

Literatura

1. https://www.python.org/
2. https://www.jetbrains.com/pycharm
3. https://www.visual-paradigm.com/
4. Jim Hugunin, Corporation for National Research Initiatives “Python and Java: The Best of Both Worlds”
5. Sebastian Nanz, Carlo A. Furia Chair of Software Engineering, Department of Computer Science, “A Comparative Study of Programming Languages in Rosetta Code,”
6. James Rumbaugh, Ivar Jacobson, Grady Booch, The Unified Modeling Language Reference Manual, Second Edition