Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/15BE08Djekic

RAZVOJ APLIKACIJE ZA DETEKTOVANjE SOFTVERSKIH OBRAZACA

DEVELOPMENT OF APPLICATION FOR DETECTION OF SOFTWARE PATTERNS

Nemanja Đekić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu demonstrirana je tehnika detektovanja softverskih obrazaca gde je implementirana aplikacija koja radi sa podacima koji predstavljaju „XMI“ reprezentaciju softverskog sistema. Aplikacija radi sa „NoSQL“ bazom podataka. Aplikacija je razvijena u „Microsoft Visual Studio“ programskom okruženju korišćenjem „C Sharp C#“ programskog jezika.*

Ključne reči: *Softverski obrasci, Algoritam, Detekcija*

Abstract – *This paper demonstrates the technique of software processing detection where an application is implemented that works with "XMI" representation of the software system. The application works with the "NoSQL" database. The application was developed in the "Microsoft Visual Studio" programming environment using the "C Sharp C #" programming language.*

Keywords: *Software Patterns, Algorithm, Detection*

1. UVOD

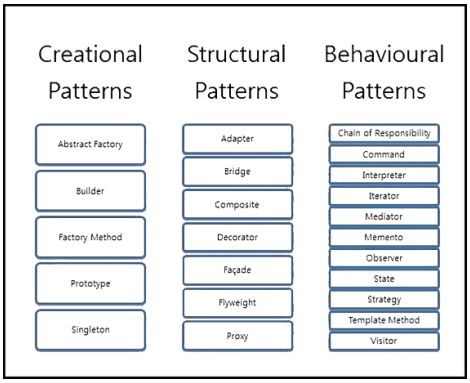
Objektno orijentisani obrasci predstavljaju dobro poznata rešenja za uobičajene probleme dizajna u datom kontekstu. Najpoznatija kolekcija objektno orijentisanih obrazaca sadržana je u knjizi „Gamma et al“.
Autori su prikupili i dokumentovali 23 uzorka obrazaca koje su predstavili kroz programske jezike „Smalltalk“ i „C++“ [1, 2]. Kroz ovaj rad biće sumirani određeni pristupi u analizi prisutnosti određenih objektno orijentisanih obrazaca u okviru raznih reprezentacija sistema ali će se analizirati pristup kao i aplikacija koja je nastala kao rezultat ovoga istraživanja.
Pristup u analizi zasniva se na višestruko redukciono strategiji baziranoj na podacima kao i meta podacima koji su nastali u procesu kreiranja klasnog dijagrama sistema.
Benefiti koji proizilaze iz upotrebe objektno orijentisanih dizajn obrazaca su višestruki pa tako iz ugla razumevanja aplikacija koje implementiraju određene obrasce možemo uočiti da obrasci pružaju informacije o rolama određenih klasa, razloge određenih veza među sastavnim delovima obrazca kao i preostalim delovima sistema.
Drugačije rečeno, otkrivanje obrazaca u okviru aplikacija predstavlja korak ka razumevanju procesa koji čine samu aplikaciju kao i dokumentaciju koja je opisuje.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio prof. dr Darko Čapko.

Shodno tome identifikacija obrazaca pruža uvid u strukturu softvera kao i uvid u mesta u softveru gde je moguće uvesti određene promene ali i proširenja. Štaviše sistem koji je dizajniran tako da koristi dobro poznate, dokumentovane i prihvaćene obrazce, pokazaće u produkciji dobra svojstva kao što su modularnost, razdvajanje odgovornosti, višestruke upotrebljivosti kao i lakoću proširenja. Pored ovoga upotreba određenih obrazaca može pomoći u davanju određenih informacija ali i naznaka o kvalitetu celokupnog sistema. Prisustvo dizajn obrazaca u rešenju treba da se reflektuje tako na sistem da izvlačenjem veza i uzorkovanjem dizajna aplikacije možemo da opravdamo izbor određenog rešenja u određenom sistemu te tako pojednostavimo izgradnju određenog konceptualnog modela sistema [3].

2. SOFTVERSKI OBRASCI

Trenutna upotreba izraza obrazac izvedena je iz pisanja arhitekte „Christopher Alexander“ koji je napisao i objavio više izdanja na temu dizajn obrazaca. Knjige su opisivale arhitekturu i urbanističko planiranje, ipak ideje su bile višestruko primenjive na više različitih disciplina pa tako i na razvoj softvera. Softverski obrasci postali su popularni i prihvatljivi sa pojavom knjige „Gamma et al“ 1995. godine, te od tada interesovanje oko obrazaca, jezika i načina na koji se obrasci implementiraju raste eksponencijalno i danas je prisutno gotovo u svakom softveru [2].
Prema klasifijaciji predloženoj u knjizi „Gamma“, obrasci mogu se klasifikovati na kreacione, strukturalne i obrasce ponašanja. Kreacioni obrasci obuhvataju stvaranja objekata, strukturalni obrasci obuhvataju klase i objekte, odnosno njihov sastav, dok se obrasci ponašanja bave načinima na koje klase raspoređuju odgovornost međusobno te proizvode rezultat [4]. Na slici 2.1 možemo videti klasifikaciju softverskih obrazaca koji se aktivno koriste u industriji.



Slika 2.1 *Objektno orijentisani obrasci [9]*

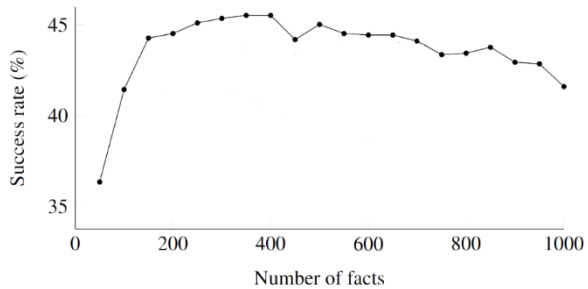
3. ALATI ZA DETEKTOVANJE OBRASACA

Gledano kroz istoriju softvera, od 1990. godine pa do danas razvila su se ali i dalje se razvijaju rešenja za detektovanje softverskih obrazaca u sistemima kako bi se olakšao pristup, promena ali i unapređenje rešenja problema određenog sistema.

U ovome radu biće opisano nekoliko rešenja koja postoje u industriji, od alata koji analiziraju „UML“ dijagrame zatim alata koji analiziraju data ograničenja sistema te na osnovu toga detektuju obrasce zatim rešenja koja pomoću meta podataka sistema daju aproksimaciju prisustva obrazaca u sistemu, do rešenja koja koriste mašinsko učenje kako bi detektovala prisustvo obrazaca u sistemu. U daljem tekstu biće dat kratak opis jednog od rešenja.

3.1 LAMBDES

“LAMBDES” (eng. Logic Analyser of Models Based on Descriptive Semantic) je prototip softverskog alata koji radi sa „UML“ modelima tako što ih prevodi u „FOL“ (eng. First order logic) sisteme te tako radi analizu i detekciju obrazaca određenim algoritmom te dobijene rezultate upoređuje sa već poznatim i fiksiranim parametrima alata koji opisuju obrasce te dalje sve dobijene podatke prosleđuje u „SPASS“ (eng. Synergetic Prover Augmenting Superposition with Sorts) koji je zapravo automatizovan sistem dokazivanja teorema te on dalje proizvodi krajnji rezultat analize.
Alat je skalabilan i veoma se lako integriše u ostale alate slične namene. Alat funkcioniše tako što radi prevođenje dijagrama, zatim određenim algoritmom pravi određene skupove pravila koji predstavljaju semantiku sistema.
Na osnovu statičke analize celokupnog sistema alat generiše skup pomoćnih konstanti koji mu pomaže u analizi. Korisnik alata je takođe u mogućnosti da navede određena ograničenja kako bi alat proizveo što bolje rešenje.
Kada su prepoznata sva pravila, konstante i ograničenja alat radi analizu i daje određene rezultate. Ukoliko korisnik alata nije zadovoljan rezultatima analize on je u mogućnosti da postavi nova ograničenja koja bi dovela do boljih rezultata analize.
Ovakav pristup detektovanju obrazaca prvi put je viđen upravo u ovome alatu, gde zapravo sam korisnik ima veoma veliku ulogu u samom radu i direktno utiče na detekciju obrazaca [5].
Na slici 3.1 prikazana je stopa uspešnosti alata u detekciji softverskih obrazaca u odnosu na broj raspoloživih podataka na osnovu kojih se vrši detekcija.

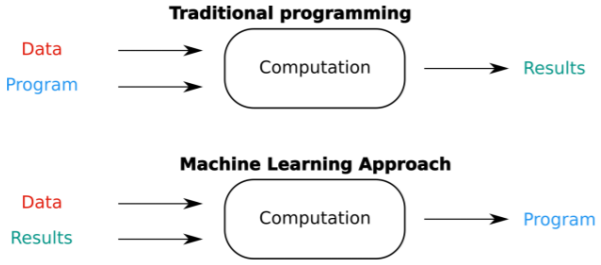


Slika 3.1 *Uspešnost „SPASS” sistema [5]*

4. MAŠINSKO UČENJE KAO ALAT ZA DETEKCIJU OBRAZACA

Obrasci softverskog dizajna su apstraktni opisi rešenja problema koji se ponavljaju u softverskoj industriji. Informacije o tome gde se koji obrazac primenjuje veoma je su korisne i važne za softversko održavanje i evoluciju. Ove informacije se obično dobijaju korišćenjem određenih alata koji su skalabilni i daju poprilično dobre rezultate ali i dalje zahtevaju da sam korisnik alata definiše određene parametre u radu. Ovo znači da postojeća rešenja za detekciju obrazaca nisu gotovo ni malo automatizovani procesi te sve informacije koje izgenerišu često budu samo jednom upotrebljene ali često i izgubljene. Upravo zbog toga, mašinsko učenje kao alat donosi revoluciju u svet detektovanja obrazaca u softverskim sistemima. Mašinsko učenje oslanja se na već postojeće podatke o softverskim obrascima, alate koji već postoje u industriji za detekciju softverskih obrazaca ali naravno i veštačke neuronske mreže koje su obučene da dobijene podatke analiziraju.

Predloženi pristup prepoznavanja dizajn obrazaca na osnovu metoda mašinskog učenja započinje prvom fazom u kojoj se smanjuje prostor za pretragu identifikovanjem skupa klasa kandidata za svaku ulogu u svakom dizajn obrascu. Zatim se u drugoj fazi za sve moguće kombinacije odnosno kandidate dizajn obrazaca proverava da li su oni zapravo valjana instanca nekog od dizajn obrazaca. Odvojena veštačka neuralna mreža „ANN“ (eng. Artificial neural network) se zatim obučava paralelno sa različitim vektorima obeležja koji dekorišu valjane instance dizajn obrazaca, pored ovoga neuralna mreža radi i sa podacima koji su nastali kao rezultat već postojećih rešenja za detektovanje obrazaca u softverskim sistemima. Kada se obrada podataka izvrši, ukoliko su neuronske mreže u procesu paralelnog obučavanja imale stopu uspešnosti veću od pedeset procenata, možemo očekivati da će alat detetektovati barem neki softverski obrazac sa velikom sigurnošću [2, 3]. Na slici 4.1 ilustrovana je razlika u pristupu između tradicionalnog programiranja i pristupa programiranju koji nudi mašinsko učenje.



Slika 4.1 *Pristup tradicionalnog programiranja i mašinskog učenja [5]*

Ono što je bila motivacija za izradu prvih alata koji su koristili neuronske mreže jesu zapravo dva povezana motiva i cilja ka kojima se težilo. Prvi je da se u potpunosti iskoristi snaga i sposobnost metoda mašinskog učenja i upotrebe neuronske mreže u rešavanju problema odnosno prepoznavanju softverskih obrazaca. Uprkos veoma poznatom i velikom uspehu neuronskih mreža u

rešavanju drugih problema, problem prepoznavanja dizajn obrazaca bio je potpuno nov u svetu neuronskih mreža. Ono za šta su se najviše do tog momenta koristile neuronske mreže jeste zapravo filtriranje lažnih pozitivnih rezultata ili smanjivanje prostora za pretragu kako bi doveli do pronalaska rešenja problema stavljenog pred njih. Drugi motiv razvoja ovakvog pristupa jeste da se u potpunosti nauče pravila i funkcije prepoznavanja instanci softverskih obrazaca u sistemima koji se već izvršavaju na nekoj mašini [3,7].

Gotovo sve tehnike koje su do tada postojale za detektovanje softverskih obrazaca zavisile su u različitoj meri od pravila i osobina izvedenih iz teorijskog opisa softverskih obrazaca što ne znači da su eksplicitno te osobine kroz implementaciju dospele u softver. Neke od tih tehnika zasnivali su se na tome da su teorijski opisi obrazaca u potpunosti prisutni u već implementiranim softverima, što naravno nije slučaj [6].

Upravo zbog ovakvih pristupa koji su kroz praksu pokazali veoma malu stopu uspeha, mašinsko učenje i neuronske mreže krenule su drugačijom putanjom. One su krenule od analize ne teorije dizajn obrazaca nego samih instanici dizajn obrazaca u okviru softvera. Ovakav pristup dao je neočekivano dobre rezultate i gotovo neuporedivo veći set podataka koji podrazumeva obeležja i pravila nad kojima su neuronske mreže mogle da rade i da se treniraju. Treniranje neuronskih mreža ovakvim pristupom pomoglo je izbegavanju problema lažnih pozitivnih poklapanja ali je donelo i podršku za veću granularnost rešenja, što postojeća rešenja gotovo nisu ni imala u vreme kada je prvi alat ovakve prirode nastajao [3,8].

5. APLIKACIJA

U okviru master rada pravljena je aplikacija radi demonstracije tehnike detektovanja softverskih obrazaca. Aplikacija radi nad podacima dobijenim modelovanjem softvera korišćenjem alata „Enterprise Architect“. Aplikacija se sastoji iz dva osnovna dela. Prvi deo aplikacije zadužen je da modeluje podatke koji se mogu naći u „XMI“ reprezentaciji sistema koji alat generiše na osnovu modela sistema odnosno dijagrama klasa sistema, dok je drugi deo aplikacije zadužen za parsiranje podataka dobijenih kroz reprezentaciju sistema, detektovanje obrazaca u okviru tih podataka na osnovu određenog algoritma kao i za generisanje koda na osnovu dobijenih podataka.

Algoritam se sastoji iz dva dela:

1. Transformacija „XMI“ reprezentacije softvera u objektnu reprezentaciju pogodnu za dalju analizu.
2. Analiza dobijenih podataka i prepoznavanje softverskih obrazaca.

Aplikacija se nalazi u okviru jednog rešenja i sastoji se iz tri zasebna dela. Prvi deo rešenja sadrži sve klase koje su zadužene za smeštanje informacija dobijenih parsiranjem „XMI“ reprezentacije softvera. Pored smeštanja informacija, pojedine klase zadužene su i za dalju separaciju izvučenih podataka. Zadatak drugog dela jeste da detaljno analizira „XMI“ reprezentaciju softvera odnosno dijagrama klasa, da detektuje prisustvo softverskih obrazaca, kao i da generiše kod na osnovu dobijenih podataka. Algoritam radi tako što prolazi kroz sve prikupljene podatke, i na osnovu informacija o imenima klasa, imenima metoda, imenima parametara metoda, imenima polja u okviru klasa nasleđivanja u okviru klasa, povratnih vrednosti, pristupnosti polja, pristupnosti klasa, pristupnosti interfejsa, vrši analizu i uz određenu logiku detektuje prisistvo softverski obrazaca. Ukoliko je aplikacija detektovala prisustvo softverskog obrasca, rezultat upisuje u bazu podataka i nastavlja dalju analizu podataka.

Kako sam algoritam prati svaku vezu između klasa, metoda, povratnih vrednosti, vrednosti parametara, preciznost algoritma nije potpuna te je pojavljivanje lažnih pozitivnih rezultata moguće. Ono što je bitno da se napomene u vezi algoritma za detektovanje obrazaca koji je implementiran u okviru ovoga rešenja jeste to da za svaki obrazac algoritam ima kolekcije potencijalnih imena polja, metoda, getera i setera koje se koriste u softverskoj industriji. Ove kolekcije nisu od primarnog značaja za rad algoritma i veoma malo utiču na ishod, ali su upotrebljene kako bi se smanjila stopa detektovanja lažnih pozitivnih rezultata. Treći deo aplikacije jeste sam korisnički interfejs koji je implementiran tako da izvrši sve potrebne validacije pre nego to sam algoritam detekcije i generisanja koda izvrši.

U daljem tekstu biće dat opis kao i pseudokod detekcije jednog od podržanih obrazaca. Obrazac uzorak, softver detektuje tako što prolazi kroz sva polja u okviru svih klasa i mapira ih. Uz mapiranje samih polja algoritam za svako polje vezuje i pravo pristupa tom polju kao i naziv. Posle mapiranja polja algoritam ponovo prolazi kroz sve podatke i mapira sve getere i setere u okviru klasa. Ukoliko se utvrdi da postoji veza između polja koje je privatno u okviru neke klase i getera u okviru te klase koji ima povratnu vrednost istu kao što je i tip polja, klasa se obeležava kao kandidat za softverski obrazac uzorak. Ukoliko pored toga sam naziv getera nosi neki od naziva koji se obično koriste kada se implementira obrazac uzorak, klasa se obeležava kao kandidat za obrazac uzorak. Procedura detektovanja može se predstaviti i pomoću pseudokoda:

```
foreach (x in fields)
if (x.Type == y || posibleNames.contains(y.Name))
        Mark as possible candidate for pattern.

foreach (x in properties)
if (x.Type == this.Type)
        Mark as possible candidate for pattern.

foreach (x in markedEntities)
if (x.MarkForReason == reasonOfDetection)
        Add to collection of detections.

Check collection for detection count and proclaim
detection if possible.
```

5.1 Testiranje softvera

U daljem tekstu biće opisana metodologija testiranja softvera koji je proizišao iz ovoga rada kao i stopa

uspešnosti koju je softver imao u detekciji softverskih obrazaca.

Testiranje softvera je izvršeno kroz par koraka a oni su:

1. Pronalaženje implementacija softverskih obrazaca u „C#“ programskom jeziku.

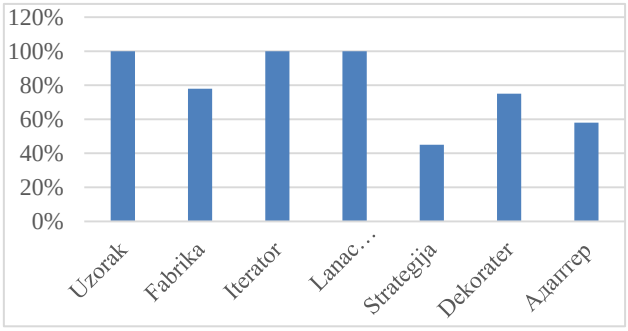
2. Prebacivanje implementacija u dijagrame klasa na osnovu kojih se radi analiza prisustva softverskih obrazaca.

3. Puštanje softvera u rad i analiza dobijenih rezultata.

U prvom koraku za svaki softverski obrazac, prikupljeno je najmanje četiri različite implementacije softverskog obrasca. Implementacije su prikupljene iz različitih izvora kako bi se eliminisala mogućnost ponavljanja. Nakon prikupljanja, svaka implementacija posebno je prebačena u dijagram klasa pomoću softvera „Enterprise Arhitect“, te je nakon toga na osnovu dijagrama klasa generisana „XMI“ reprezentacija dijagrama.

Nakon generisanja svaka reprezentacija posebno je upotrebljena kao izvor podataka softvera za detektovanje softverskih obrazaca. U tabeli 1 prikazani su rezultati testiranja. Ono što može da se zaključi jeste da softver ima veoma veliku stopu uspešnosti u detekciji softverskih obrazaca čija implementacija nije direktno zavisna od implementacije tela metoda, već se detekcija može odraditi na osnovu tipova povratnih vrednosti, parametara metoda i polja ali i na osnovu imena koja su upotrebljena u implementaciji kao i ostvarenim vezama između samih klasa, kao što je to slučaj za obrazac uzorak.

Tabela 1: *Rezultati testiranja*



6. ZAKLJUČAK

Sa pojavom objektno orijentisanog pristupa u softveru, softverski obrasci napravili su pravu pometnju u razvoju softvera. Klasične strukture koje su podrazumevale jedan ogroman fajl sa puno odgovornosti gotovo su nestale. Novi pristup koji su doneli softverski obrasci razdvojio je odgovornosti, povećao je skalabilnost, doneo je bolje performanse.

Dalji razvoj na ovome polju gotovo je neminovan i sa pojavom novih tehnologija prisustvo softverskih obrazaca je zagarantovano.

Za potrebe ovog rada napravljena je implementacija jedne od tehnika detekcije softverskih obrazaca.

Cilj implementacije bila je demonstracija rada tehnike u industriji kroz manji primer, nad relativno manjem obimu podataka. Dalji pravci u istraživanju u okviru ovoga rada jesu dublje razumevanje integracije mašinskog učenja sa detekcijom softverskih obrazaca kao i softverski ali i hardverski zahtevi koje ta integracija donosi sa sobom.

7. LITERATURA

[1] G Guéhéneuc, Y. G., & Antoniol, G. Demima, A multilayered approach for design pattern identification, *IEEE transactions on software engineering* 34.5: 667-68, 2008.

[2] Kramer, C., & Prechelt, L., Design recovery by automated search for structural design patterns in object-oriented software, *Proceedings of WCRE'96: 4rd Working Conference on Reverse Engineering*, 1996.

[3] Fabry, J., & Mens, T., Language-independent detection of object-oriented design patterns, 2004.

[4] Srinivasan, S. Design patterns in object-oriented frameworks. Computer, 1999.

[5] Zhu, H., Bayley, I., Shan, L., & Amphlett, R., Tool support for design pattern recognition at model level, *2009 33rd Annual IEEE International Computer Software and Applications Conference*, 2009.

[6] Jing Dong, J., Sun, Y., & Zhao, Y., Design pattern detection by template matching, *Proceedings of the 2008 ACM symposium on Applied computing*, 2008.

[7] Afloz Chakure, Introduction to Machine Learning, 2019

[8] Prakash, N., Manconi, A., & Loew, S., Mapping landslides on EO data: Performance of deep learning models vs. traditional machine learning models, *Remote Sensing* 12.3: 346, 2020.

[9] Fowler, M., Patterns [software patterns], 2003

Kratka biografija:

Nemanja Đekić rođen je u Subotici 1995. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Razvoj aplikacije za detektovanje softverskih obrazaca odbranio je 2021.god.

kontakt:
nemanja.djekic.1995.2019@gmail.com