Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 007:004.056
DOI: https://doi.org/10.24867/16BE17Sarenac

IDENTIFIKACIJA KOHEZIVNIH CJELINA KLASE

IDENTIFYING COHESIVE PARTS OF A CLASS

Balša Šarenac, *Fakultet tehničkih nauka, Novi Sad*

Oblast – RAČUNARSTVO I AUTOMATIKA

Kratak sadržaj – *U radu je opisan algoritam pronalaska kohezivnih cjelina unutar klase. Algoritam nudi korisniku moguće načine refaktorisanja klase na dvije manje visoko kohezivne klase.*

Ključne reči: *OOP, kohezija čist kod, refaktorisanje*

Abstract – *The paper describes an algorithm that finds highly cohesive parts of a class. As a result, it offers user possible ways of refactoring given class to create two smaller more cohesive classes.*

Keywords: *OOP, cohesion, clean code, refactoring*

1. UVOD

Čist kod je jedna od najbitnijih stvari koja čine kvalitetne softverske projekte. Čist kod se može definisati kao jednostavan, čitak, razumljiv i lak za testiranje. Iako se lako može prepoznati kad je neki kod čist, pisanje čistog koda nije lako, pogotovo ne za početnike, jer je to vještina koja se uči s vremenom [1]. Upravo to je razlog zašto bi alati koji pomažu pri pisanju čistog koda bili korisni početnicima.

Jedan od načina za mjerenje kvaliteta koda je kohezija. Kohezija označava stepen povezanosti komponenti unutar nekog modula. Moduli sa visokom kohezijom su obično lakši za razvoj, održavanje, testiranje i ponovnu upotrebu [2]. U kontekstu objektno-orjentisanog programiranja, kohezija se može posmatrati na nivou klase. Tada se kohezija ogleda kao stepen povezanosti atributa i metoda. Tako je klasa visoko kohezivna ako sve metode interaguju sa svim atributima.

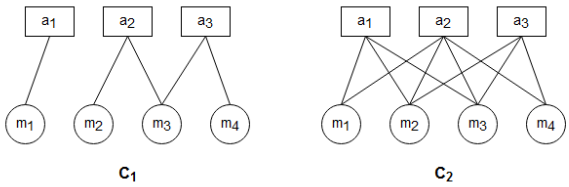
Ovaj rad predlaže rješenje koje će identifikovati kohezivne cjeline unutar neke klase. Dato rješenje nudi programeru više mogućnosti podjele klase na dvije kohezivne cjeline sa svim potrebnim informacijama za refaktorisanje.

U poglavlju 2 je kohezija detaljno opisana i neki ključni koncepti koji su usko povezani sa kohezijom. U poglavlju 3 je opisan dizajn rješenja. U poglavlju 4 je odrađena analiza implementacije, a u poglavlju 5 je zaključak.

2. KOHEZIJA

Fokus ovog rada je na analizi i računanju kohezije klasa u objektno-orjentisanom programiranju (OOP). Kohezija se može posmatrati na različitim nivoima apstrakcije. Tako se u OOP, pored kohezije klasa, može posmatrati kohezija klasa i fajlova unutar paketa.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Nikola Luburić, docent.



Slika 2.1 *Primjeri klasa predstavljenih kao grafovi*

Na primjer klase c_1 i c_2 prikazane na slici 2.1. Klase su predstavljene u obliku grafa, gdje su pravougaonici sa oznakama a_i atributi, a krugovi sa oznakama m_j su metode. Linija koja povezuje metodu sa atributom predstavlja neku interakciju (čitanje ili pisanje) metode i atributa. Na prvi pogled je jasno da je klasa c_2 kohezivnija od klase c_1.

Jedan od razloga zbog kojeg su moduli sa visokom kohezijom poželjni jeste što oni obično zadovoljavaju princip jedne odgovornosti (engl. *Single Responsibility Principle* – SRP). Jedna od definicija ovog principa je: „Okupi stvari koje se mijenjaju iz istih razloga. Razdvoji stvari koje se mijenjaju iz različitih razloga.“, što se može posmatrati i kao drugi način da se opiše kohezija. Stvari koje se mijenjaju iz istih razloga treba da imaju visoku koheziju.

Da bi se procjena stepena kohezije nekog modula automatizovala i učinila objektivnijom, pojavila se potreba za kohezivnim metrikama.

2.1. Metrike za mjerenje kohezije klase

Postoji veliki broj ovih metrika. Kako mjere kohezivnost na različite načine, dijele se po u kom mogu da računaju koheziju:

- Metrike bazirane na interfejsu – koheziju računaju tokom faze dizajna. Ove metrike koriste podatke iz definicija metoda (npr. broj parametara i tipovi parametara).
- Metrike bazirane na kodu – koheziju računaju u fazi razvoja. Ove metrike koriste podatke iz koda da bi računali koheziju (npr. pristupe metoda atributima, sadržaj komentara, itd.). Imaju dvije podgrupe:
  - Semantičke metrike – računaju koheziju na osnovu nekih koncepata izvornog koda (npr. imena identifikatora i sadržaj komentara).
  - Strukturalne metrike – koriste strukturalne podatke (npr. pristupe metoda atributima).

Generalno gledano bolje rezultate daju metrike bazirane na kodu (konkretno strukturalne metrike), jer imaju više informacija i meta-podataka na raspolaganju [3].

2.2. Poželjna svojstva

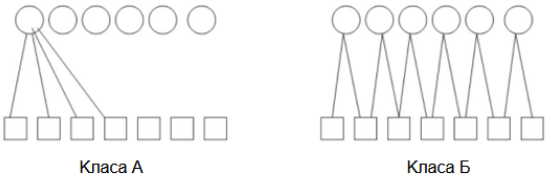
Sa velikim brojem metrika javila se i potreba za načinom da se mjeri kako teorijska tako i empirijska validnost metrika. Četiri poželjna svojstva koja su definisali Briand at al. [4] su postala standard za teorijsku validaciju:

1. Normalizacija i nenegativnost – metrike treba da budu normalizovane na interval od 0 do 1.
2. *Null* i maksimalna vrijednost – metrika treba da ima definisanu maksimalnu i *null* vrijednost.
3. Monotonost – dodavanjem konekcija unutar modula ne bi trebalo da smanji koheziju.
4. Kohezivni moduli – Kohezija modula dobijenog spajanjem dva nepovezana modula ne bi trebalo da bude veća od kohezije dva originalna modula.

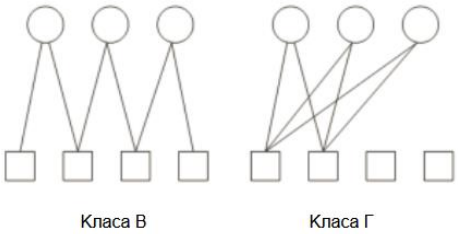
Metrike koje zadovoljavaju ova svojstva su dobro definisane [3]. Zadovoljavanje ovih svojstava je prvi korak ka definisanju kvalitetnih metrika.

2.3. LDA problem

Pored četiri poželjna svojstva, ne uzimanje u obzir interakcija unutar modula (engl. *Lack of discrimination anomaly* – LDA [5]) je još jedno svojstvo koje je bitno za kohezivne metrike. Ovo svojstvo je nepoželjno u smislu da metrike ne treba da ga ispoljavaju. LDA se ispoljava kod metrika koje ne uzimaju u obzir interakcije metoda i atributa unutar klase ili zanemaruju njihovu raspodjelu. Na slikama 2.2. i 2.3 su primjeri klasa za koje metrike koje ispoljavaju LDA ne mogu tačno odrediti koheziju.



Slika 2.2. K*lase za koje metrike ne mogu tačno odrediti koheziju ukoliko ne uzimaju u obzir veze između metoda i atributa*



Slika 2.3 *Klase za koje metrike ne mogu tačno odrediti koheziju ukoliko ne uzimaju u obzir raspored veza*

2.4 Diskusija

Metrike koje zadovoljavaju poželjna svojstva i ne ispoljavaju LDA problem generalno daju bolje rezultate od metrika koje ih ne zadovoljavaju.

Metrike za računanje kohezije su zgodne jer za kratko vrijeme daju povratnu informaciju o kodu. Međutim, velika mana strukturalnih metrika je što ne koriste i ne uračunavaju semantiku neke klase. Semantičke metrike pokušavaju da uračunaju semantiku, međutim da bi ove metrike dale dobre rezultate potrebno je da imena identifikatora budu smislena i pažljivo odabrana, tj. da su komentari smisleno napisani. Postoje neki radovi koji su pokušali da kombinuju semantičke i strukturalne metrike da dobiju bolje rezultate [6].

3. DIZAJN RJEŠENJA

Osnovna ideja je da se pomoću kohezivnih metrika odrede kohezivne cjeline unutar klase i ponude korisniku kao opcije za refaktorisanje.

Od istraženih metrika ICMBC [2] se izdvojio kao dobar kandidat. Ova metrika rekurzivno uklanja interakcije između metoda i atributa da bi dekomponovala klasu. Na osnovu [3] je utvrđeno da ICBMC zadovoljava sva 4 poželjna svojstva i ne ispoljava LDA problem. Iz navedenih razloga je ova metrika uzeta kao početna tačka implementacije.

Za razumijevanje ICBMC metrike potrebno je razumjeti metriku na osnovu koje je definisana ICBMC, a to je CBMC [7].

3.1. CBMC

Za početak će se uvesti par definicija koje će se referencirati u ostatku rada.

Definicija 3.1 Metoda klase C je **specijalna** ako je ili metoda pristupa (engl. *accessor*), metoda delegacije, konstruktor ili destruktor. Metode koje nisu specijalne su **normalne**.

Definicija 3.2 **Referentni graf** za klasu C je usmjereni graf $G = (N, A)$ gdje je:

- $N = Nv \cup Nm$, gdje su Nv i Nm skup svih atributa i skup svih normalnih metoda klase C
- $A = \{(n, v) \mid m\ čita\ ili\ piše\ v, m \in Nm, v \in N\_v\}$

Definicija 3.3 Skup ljepljivih metoda je minimalan podskup metoda bez kojih je referentni graf diskonektovan.

Definicija 3.4 Kohezija referentnog grafa predstavlja umnožak faktora povezanosti i faktora strukture. Faktor povezanosti je količnik broja ljepljivih metoda i ukupnog broja metoda. Faktor strukture je srednja vrijednost kohezije podgrafova nastalih dekompozicijom referentnog grafa.

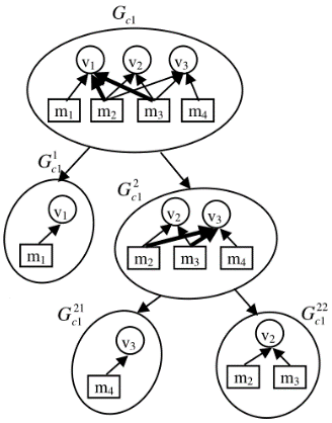
Definicija 3.5 Elementarne komponente su grafovi sa minimalno dva čvora, gdje je maksimalno može biti tačno jedana metoda ili jedan atribut.

3.2 ICBMC

ICBMC (*Improved* CBMC) metrika dekompoziciju klase vrši uklanjanjem veza između metoda i atributa, kao što slika 3.4 ilustruje.

Definicija 3.6 Skup za sječenje (*Cut set*) referentnog grafa je podskup skupa veza između metoda i atributa takav da se njihovim uklanjanjem referentni graf može podijeliti na dva manja grafa i da je taj set je minimalan (svaki njegov nadskup ne zadovoljava prvi uslov).

Definicija 3.7 Po ICBMC kohezija za referentni graf se računa kao proizvod odnosa broja veza u skupu za sječenje i ukupnog broja veza sa prosječnom vrijednošću kohezije podgrafova.



Slika 3.4. *Primjer dekompozicije klase po ICBMC*

3.2.1 Diskusija

Iako ICBMC ispravlja neke nedostatke CBMC i sama ima neke nedostatke. Naime, tokom implementacije se ispostavilo da ICBMC ipak ne zadovoljava svojstvo monotonosti [8].

Vrijeme izvršavanja je dugačko jer se na svakom nivou drveta prolazi kroz sve kombinacije veza bez ponavljanja da bi se odredio podskup veza za sječenja. Dakle to je rekurzivni $O(2^n)$ algoritam. Algoritam staje kada se pronađu prvi minimalan skup veza za sječenje, međutim dekompozicija se vrši sve do elementarnih komponenti.

Zbog svega navedenog je odlučeno da se urade određene modifikacije.

Odbačena je rekurzivna podjela grafa na podgrafove. Zadržana je osnovna ideja da se klasa pokuša izdijeliti uklanjanjem veza između atributa i metoda.

Coh [8] je uzeta kao kohezivna metrika za određivanje kohezije podgrafova, jer je za izračunavanje ICBMC potreban rekurzivni silazak.

Dalje je smanjen broj maksimalnog podskupa veza za sječenje. Inicijalno su pravljene sve kombinacije bez ponavljanja svih veza u grafu. Empirijski je utvrđeno da većina tih podskupova daju nevalidne klase ili su nadskupovi nekog drugog rješenja. Iz tog razloga je maksimalan broj veza za sječenje ograničen na: $\frac{n}{2}+1$, gdje je n ukupan broj veza.

Sa ovim modifikacijama kompleksnost algoritma je spala na $O(2^{\frac{n}{2}+1})$.

ICBMC i CBMC izbacuju sve specijalne metode iz računice, jer one interaguju sa ograničenim brojem atributa. Da taj ograničeni broj atributa ne bi narušio koheziju, specijalne metode se ne uzimaju u obzir prilikom računanja. U radu [9] je zaključeno da uključivanje specijalnih metoda prilikom predikcije klasa kojima je potrebno refaktorisanje znatno smanjuje rezultate predikcije i predloženo je da se metode pristupa i delegacije izbace, a konstruktori zadrže. Suprotno ovoj preporuci, odlučeno je da se konstruktori izbace. Konstruktori služe za inicijalizaciju atributa, samim tim nema potrebe da se uklanjaju interakcije konstruktora sa atributima. Nakon obavljenog refaktorisanja, početni konstruktor se može podijeliti na dva djela ukoliko za tim ima potrebe. Metode pristupa (čitanja i pisanja) su izbačene, a metode delegacije su ubačene. Razlog zbog kog su metode delegacije zadržane je jer nije bilo adekvatnog načina da se utvrdi da li je metoda delegator ili metoda od jedne linije koja radi neke bitne kalkulacije.

Urađeno je i poređenje vremena izvršavanja između algoritma koji koristi ICBMC i modifikovanog algoritma. Poređenje je rađeno na mašini sa Intel i7-4702Q, 2.2GHz i 16GB RAM. Vrijeme izvršavanja starog algoritma je oko 3.9 sekundi na 100 izvršavanja, dok je modifikovani algoritam trajao oko 850 milisekundi, što je 80% brže. Test je odrađen na klasi od 3 polja i 4 metode, sa 10 interakcija.

4. DIZAJN ALGORITMA

Implementacija predloženog rješenja je odrađena u sklopu platforme Clean CaDET. Ova platforma je osmišljena da pomogne programerima da pišu kvalitetniji softver. Algoritam će se koristiti prilikom evaluaciji klasa, ali kao i pomoć pri pisanju čistijeg koda.

4.1. Specifičnosti implementacije

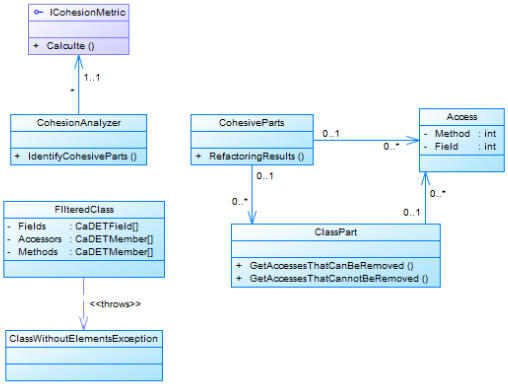
Implementacija je rađena u C# programskom jeziku. C# je specifičan jer pored klasičnih polja klasa može da sadrži i polja koja ujedno definišu u metoda pristupa i mutacije (engl. *Accessors*). Pritom je moguće odvojeno definisati polje i *accessor*-a za to polje. Zato je odlučeno da se uključuje samo *accessor*-i koji definišu osnovne metode pristupa i manipulacije. U nastavku će se pod pojmom „polja“ smatrati unija polja i ovih *accessor*-a.

4.2. Opis strukture rješenja

Na dijagramu na slici 4.1 je prikazan dijagram klasa predloženog rješenja.

Sa dijagrama na slici 4.1 se mogu uočiti iduće klase:

- *Access* – klasa koja prestavlja interakciju (čitanje ili pisanje) metode i atributa
- *ClassPart* – klasa koja je omotač za skup interakcija. Predstavlja apstrakciju klase kao skupa interakcija koje se nalaze u njoj.
- *CohesiveParts* –klasa u koju se smještaju rezultati. Sadrži listu interakcija koje treba da se izbace da bi se klasa podijelila, kao i *ClassPart* apstrakcije klasa koje će se dobiti.
- *ICohesionMetric* – Interfejs za računanje kohezivnosti nekog dijela klase (*ClassPart*-a).
- *FilteredClass* – filtrira i validira polja i metoda početne klase.
- *CohesionAnalyzer* – pronalazi kohezivne cjeline.



Slika 4.1 *Dijagram klasa predloženog rješenja*

4.3. Opis ponašanja

Algoritam prvo kreiranja *FilteredClass* instancu na osnovu *CaDETClass* instance. Klasa je validna ako ima barem jedno polje i barem jednu metodu. Idući korak je izbacivanje svih polja/metoda koja se ne koriste od strane ni jedne metode/polja. Isfiltrirana polja i metode se pretvaraju u nizove. Na osnovu njih se kasnije lako mogu namapirati interakcije na odgovarajuće metode i polja. Na osnovu *FilteredClass* instance se formira početni *ClassPart* koji sadrži sve interakcije metoda i polja.

Dalje se na osnovu početnog skupa interakcija prave manji podskupovi interakcija koji se mogu izbaciti iz klase. Oni predstavljaju kandidate pomoću kojih će se početna klasa dekomponovati.

Za svaki od podskupova se pokuša dekomponovati klasa i pronaći kohezivni dijelovi. Pronalazak kohezvinih dijelova se radi tako što se izbaci podskup interakcija i pronalaze dijelovi klase. Dijelovi se traže tako što krene od jedne interakcije i dodaju se sve interakcije povezane sa njom. Pošto interakciju predstavlja veza metode i polja, povezane interakcije su one koje sadrže ili istu metodu ili isto polje. Ukoliko se pokupe sve interakcije klasa nije diskonektovana. Ukoliko se ne pokupe sve interakcije, iz početnog skupa se izbace sve pronađene i stave se u listu za povratnu vrijednost i algoritam se opet izvršava. Pošto je ograničeno da se klasa dijeli na tačno dvije manje klase, uzimaju se samo rezultati koji su pronašli tačno dva dijela klase.

Iz rješenja se izbace sva ona koja su nadskupovi nekog drugog rješenja. Razlog je što će i svaki od tih nadskupova da podjeli klasu, ali će pri tom da dodatno ukloni još neke veze. Nakon što se isprobaju svi podskupovi kao rezultat se vrati lista svih mogućih uspješnih podjela klase. Svaku podjelu čini lista interakcija koje je potrebno ukloniti i lista novodobijenih klasa u obliku *ClassPart*-a. Lista svih uspješnih podjela klase se dodatno isfiltrira tako da u njoj ostanu samo rezultati koji sadrže visoko kohezivne cjeline.

5. ZAKLJUČAK

U ovom radu je opisan algoritam pronalaska kohezivnih cjelina klase. Dat je uvod u kohezivne metrike, kao i svojstva koja bi te metrike trebalo da zadovolje. Dalje je opisan dizajn algoritma od odabira metrike do svih modifikacija koje su primjenjene da se optimizuje vrijeme izvršavanja i kvalitet rezultata. Na kraju je kratko opisano ponašanja algoritma.

Tačke proširenja uključuju eksperimentisanje sa raznim kohezivnim metrikama ili grupama metrika koje bi dale bolje rezultate pri selekciji. Uključivanje semantičkih metrika potencijalno može povećati kvalitet rezultata.

6. LITERATURA

[1] R. C. Martin, Ур., *Clean code: a handbook of agile software craftsmanship*. Upper Saddle River, NJ: Prentice Hall, 2009.

[2] Yuming Zhou, Baowen Xu, Jianjun Zhao, и Hongji Yang, „ICBMC: an improved cohesion measure for classes“, у *International Conference on Software Maintenance, 2002. Proceedings.*, Montreal, Que., Canada, 2002, doi: 10.1109/ICSM.2002.1167746.

[3] H. Izadkhah и M. Hooshyar, „Class Cohesion Metrics for Software Engineering: A Critical Review“,

[4] L. C. Briand, S. Morasca, и V. R. Basili, „Property-based software engineering measurement“, *IEEE Trans. Softw. Eng.*, doi: 10.1109/32.481535.

[5] J. Al Dallal, „Measuring the Discriminative Power of Object-Oriented Class Cohesion Metrics“, *IEEE Trans. Softw. Eng.*, Нов. 2011, doi: 10.1109/TSE.2010.97.

[6] A. Marcus, D. Poshyvanyk, и R. Ferenc, „Using the Conceptual Cohesion of Classes for Fault Prediction in Object-Oriented Systems“, *IEEE Trans. Softw. Eng.*, Март 2008, doi: 10.1109/TSE.2007.70768.

[7] Heung Seok Chae и Yong Rae Kwon, „A cohesion measure for classes in object-oriented systems“, у *Proceedings Fifth International Software Metrics Symposium. Metrics (Cat. No.98TB100262)*, Bethesda, MD, USA, 1998, doi: 10.1109/METRIC.1998.731241.

[8] L. C. Briand, J. W. Daly, и J. Wust, „A unified framework for cohesion measurement in object-oriented systems“, у *Proceedings Fourth International Software Metrics Symposium*, Albuquerque, NM, USA, 1997, doi: 10.1109/METRIC.1997.637164.

[9] J. Al Dallal, „The impact of accounting for special methods in the measurement of object-oriented class cohesion on refactoring and fault prediction activities“, *J. Syst. Softw.*, Maj 2012, doi: 10.1016/j.jss.2011.12.006.

Kratka biografija:

Balša Šarenac je rođen 7. juna 1997. godine u Trebinju, Republika Srpska. Godine 2016. upisao je Fakultet Tehničkih nauka u Novom Sadu smer Računarstvo i automatika. 2020. godine je diplomirao sa osnovnih akademskih studija i iste godine upisuje Master studije na smeru Računarstvo i automatika.