UNIVERZITET U SARAJEVU

ELEKTROTEHNIČKI FAKULTET SARAJEVO

DOMAĆA ZADAĆA 1

- samostalan rad -

VERIFIKACIJA I VALIDACIJA SOFTVERA

Student: Mašović Haris

Indeks: 17993

Odsjek: Računarstvo i Informatika

Datum:

03.11.2018

Potpis:

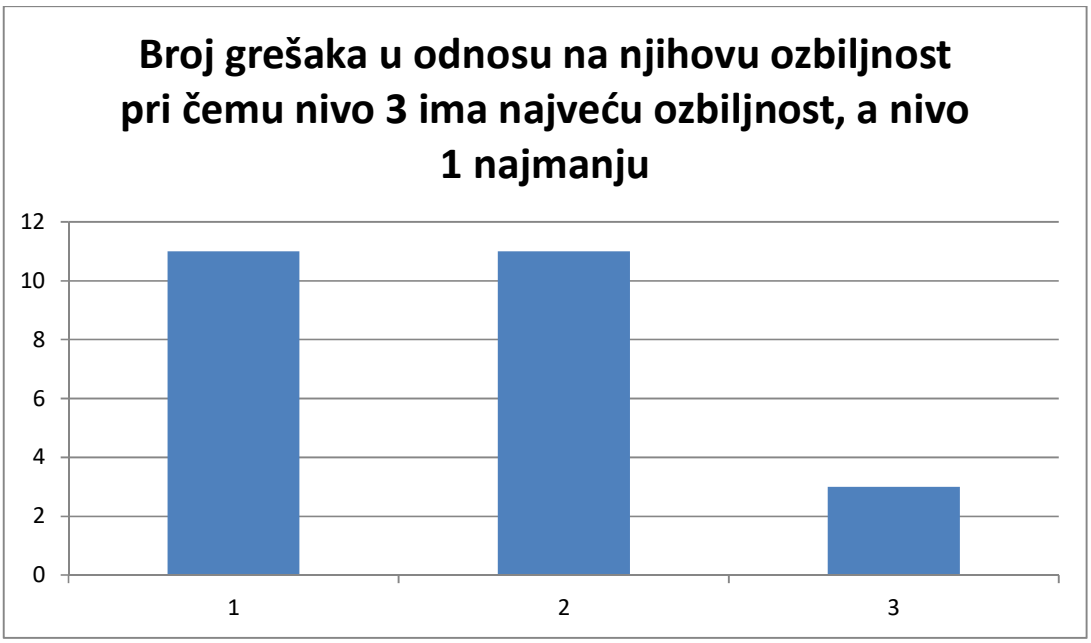
_____

Tema: Inspekcija

Zadatak b)

„Potrebno je da svaki član tima izabere jedan od statističkih alata koji nisu korišteni na vježbama i napravi pravi analizu grešaka. Svaki član mora imati drugi alat. Bodovanje je pojedinačno.“

Za rješenje ovog zadatka iskoristio sam histogram pri čemu sam prestavio broj grešaka na osnovu ozbiljnosti tih grešaka. X osa predstavlja nivo ozbiljnosti (1 – 2 – 3), a Y osa predstavlja broj grešaka na osnovu ozbiljnosti.



Ukupan broj grešaka u poslanom grupnom izvještaju je 25, što potvrđuje statistiku histograma.

Tema: Metrike

Zadatak b)

„Potrebno je da svaki član tima izabere dodatnu metriku koja nije korištena na vježbama i koju ne koristi nijedan drugi član tima, objasni je i pronađe metodu u kodu (kod koji je korišten za inspekcije) koja ima najbolju/najgoru vrijednost za ovu metriku. Bodovanje je pojedinačno.“

Za rješenje ovog zadatka iskoristio **McClure’s Complexity metriku.** Ova metrika računa kompleksnost modula kao: Complexity = C + V.
Pri čemu je C-broj komparacija u modulu, V je broj kontrolnih varijabli referenciranih u modulu.

Metrika je slična sa McCabe metrikom ali mjeri kompleksnost odluka (decisional complexity) u odnosu na kontrolne varijable.

Posmatrajmo kod sa drive-a G2/Metrike/v1/AutoRentApp.

- Najbolja vrijednost za ovu metodu je očigledno ona koja ima najmanju vrijednost pri čemu je ograničena odozdo tj. >= 0. Takva metoda je `VratiCijenuKaucije` unutar klase Klijent. Broj komparacija u ovoj metodi je jednak 0 i broj kontrolnih varijabli je jednak 0, samim tim je kompleksnost ove metode po McClure je jednaka 0.

- Za najgoru vrijednost ove metode možemo uzeti metodu `KojijeDan` iz klase AutoRentShop.cs. Broj komparacija u ovoj metodi je jednak 7 (ako uzmemo najgori slučaj tj. da je `case DayOfWeek.Sunday`) i broj kontrolnih varijabli je jednak 1, odnosno varijabla `dan`. Shodno tome kompleksnost ove metode po McClure je jednaka 8.

Napomena: Posmatrao sam skup metoda u kojima je jasno fixno unaprijed definisan broj provjera.
Mogao sam uzeti metodu kao što je recimo `VratiVozilo` unutar klase AutoRentShop.cs gdje bi broj komparacija bio zavisan od veličine liste odnosno određen dinamički, pa bi vrijednost ove metode bila n + 2, gdje n predstavlja broj poređenja unutar petlji, a 2 predstavlja broj kontrolnih varijabli.

Tema: Code Tuning

Zadatak b)

„Potrebno je da svaki član tima izabere dodatnu code tuning tehniku koja nije korištena na vježbama i koju ne koristi nijedan drugi član tima, objasni je i pronađe metodu u kodu (kod koji je korišten za inspekcije) pogodno za primjenu odabrane tehnike. Obrazložiti dobijene rezultate nakon primjene code tuning tehnike“

Odabrao sam unrolling code tuning metriku i metodu `DobaviIznajmljenaVozila` iz klase AutoRentShop.cs

Unrollong (odmotavanje) propagira da se što je moguće više posla uradi u okviru jedne iteracije petlje. Povećava broj linija unutar jedne iteracije što omogućava rješavanje više zavisnosti pri hardverskom/kompajlerskom raspoređivanju instrukcija, čime dobijamo bolje performanse.

Kod main-a:

```
static void Main(string[] args)
{
    AutoRentShop shop = new AutoRentShop();
    Klijent klijent = new Klijent("Haris", "Mašović", DateTime.Now);

    for(int i=100; i< 700; ++i)
    {
        shop.RegistrujNovoVozilo(new TeretnoVozilo("nesto", "nesto2", "1A2B3C4567G111" + i.ToString(), 16));
        shop.IznajmiAuto(klijent, "1A2B3C4567G111" + i.ToString(), DateTime.Now);
    }
    ;
    List<MotornoVozilo> rez = new List<MotornoVozilo>();
    for(int i=0; i < 1000; ++i) rez = shop.DobaviIznajmljenaVozila(klijent);
    ;
    Console.WriteLine("Broj iznajmljenih vozila: " + rez.Count);
    Console.ReadLine();
}
```

Kod prije code tuninga:

```
public List<MotornoVozilo> DobaviIznajmljenaVozila(Klijent k)
{
    List<MotornoVozilo> rez = new List<MotornoVozilo>();


    foreach (Tuple<Klijent, MotornoVozilo, DateTime> x in iznajmljenaVozila)
        if (x.Item1.Id.Equals(k.Id))
            rez.Add(x.Item2);

    return rez;
}
```

Kod poslije code tuninga:

```
public List<MotornoVozilo> DobaviIznajmljenaVozila(Klijent k)
{
    List<MotornoVozilo> rez = new List<MotornoVozilo>();

    int i = 0;
    int broj = iznajmljenaVozila.Count;
    for (; i < broj - 1; i += 2)
    {
        if (iznajmljenaVozila[i].Item1.Id.Equals(k.Id)) rez.Add(iznajmljenaVozila[i].Item2);
        if (iznajmljenaVozila[i + 1].Item1.Id.Equals(k.Id)) rez.Add(iznajmljenaVozila[i + 1].Item2);
    }
    if ((broj % 2 != 0) && iznajmljenaVozila[i].Item1.Id.Equals(k.Id)) rez.Add(iznajmljenaVozila[i].Item2);

    return rez;
}
```

Zaključak:

Trajanje izvršavanja koda prije i poslije vršenje *tuning*-a: 197.507 ms, 95.32ms

Korištenje memorije prije i poslije vršenja *tuning*-a: 13.1MB, 13.8MB

Korištenje procesorske snage prije i poslije vršenja *tuning*-a: 25 %, 25%

** Vidimo da se vrijeme izvršavanja smanjilo što je očekivano, veličina korištenje memorije povećana što je također očekivano, iskorištenost procesora je ostala ista. Rezultat code tuninga je pozitivan.