Laboratorijska vježba br. 1 **Izvještaj o Inspekciji Koda**

Check liste inspekcije koda

Označiti stavke na listama za inspekciju koje su ispunjene, nakon vršenja inspekcije koda. Za stavke koje se ne označe potrebno je navesti detaljne informacije o greškama u nastavku.

Inspekcija strukture programskog rješenja

Ova lista stavlja fokus na potencijalne probleme u strukturi programskog koda, analizi koda na visokom nivou i poštovanju standarda.

☒ Kod je napisan u skladu sa važećim standardima kodiranja.
☒ Stil kodiranja je konzistentan u cijelom programskom rješenju. ☒
Kod je ispravno formatiran.
☐ U kodu nema funkcija koje se ne pozivaju ni na jednom mjestu.
☒ Nema nedostižnih linija koda.
☒ Nema bespotrebnog implementiranja funkcija koje mogu biti zamijenjene postojećim bibliotekama.
☐ U kodu nema ponavljanja koje može biti zamijenjeno jedinstvenom funkcijom.
☒ Memorija se koristi na efikasan način.
☒ Nema korištenja *magičnih brojeva* i konstanti bez korištenja varijabli.
☒ Nema previše dugih i kompleksnih blokova koda.

Inspekcija dizajna programskog rješenja

Ova lista stavlja fokus na potencijalne probleme u poštovanju objektno-orijentisanih principa, SOLID principa i dizajn pattern-a u okviru programskog rješenja.

☒ Svaka klasa ima malu kompleksnost i jedan tip operacija i zaduženja.
☒ Klase su prilagodljive budućim promjenama.
☒ Svi objekti izvedenih klasa zamjenjivi su svojim osnovnim klasama. ☒
Interfejsi su jednostavni, s malim brojem funkcija.
☒ Dubina nasljeđivanja nije velika.
☒ Klijent može jednostavno pristupati objektima kontejnerskih klasa, bez potrebe definisanja detalja gradivnih dijelova klase.
☒ U slučaju potrebe ponovnog korištenja većeg broja istih objekata, objekti se ne instanciraju više puta.
☒ Instanciranje kontejnerske klase vrši se samo jednom.
☒ Sigurnost aplikacije osigurana je putem *proxy*-a.

Inspekcija varijabli i izraza programskog rješenja

Ova lista stavlja fokus na potencijalne probleme u strukturi koda na visokom nivou, uključujući varijable i izraze u kodu.

☒ Sve varijable imaju imena koja odgovaraju njihovoj namjeni.
☐ Koristi se jedan stil imenovanja varijabli.
☐ Nema varijabli koje se ne koriste.
☒ Nema neosiguranih potencijalnih dijeljenja s nulom.
☒ Operator = ne koristi se u logičkim izrazima.

Inspekcija petlji i grananja programskog rješenja

Ova lista stavlja fokus na potencijalne probleme u petljama i grananjima u kodu.

☒ Nema praznih niti nedostižnih blokova koda.
☒ U *if* blokovima testiraju se češći scenariji.
☒ Svi *switch* iskazi imaju definisan *default* slučaj.
☒ Sve petlje imaju uslov završetka.
☒ Nema velikog broja gniježdenja petlji.
☒ U petljama nema koda koji se može izvršiti izvan petlje.

Inspekcija memorijskih operacija programskog rješenja

Ova lista stavlja fokus na potencijalne probleme u korištenju memorije te konekciji s bazama podataka, vanjskim uređajima i korištenjem file-ova u kodu.

☒ Sve varijable koje koriste indeksiranje su inicijalizirane prije korištenja.
☒ Sva alocirana memorija dealocira se prije završetka izvršavanja.
☒ Pri radu s vanjskim uređajima, postoji provjera za *timeou0t*.
☒ Prije pokušaja modificiranja *file*-ova, provjerava se da li oni postoje. ☒
Nakon završetka transakcije, konekcija s bazom podataka se uvijek zatvara.

Inspekcija dokumentacije programskog rješenja

Ova lista stavlja fokus na potencijalne probleme u razumljivosti i jednostavnosti dokumentovanja koda.

☐ Svi kompleksni dijelovi koda posjeduju komentare.
☒ Dijelovi koda podijeljeni su u regije. ☒
Metode klasa imaju svoje opise.
☒ U cijelom rješenju koristi se jedan stil komentarisanja koda.

Informacije o timu koji vrši inspekciju koda

Popuniti informacije o članovima tima koji vrši inspekciju.

Ime i prezime, broj indexa: Berin Mašović, 111-ST
Zaduženje: Klasa algoritma BubbleSort i CombSort
Predmet inspekcije: petlje, grananje i dokumentacija programskog rješenja

Ime i prezime, broj indexa: Harun Kološ, 105-ST
Zaduženje: Klasa algoritma ShellSort i CocktailSort
Predmet inspekcije: memorijskih operacija, varijabli i izraza programskog rješenja

Ime i prezime, broj indexa: Alen Mehanović, 103-ST
Zaduženje: Klasa algoritma SelectionSort i InsertionSort
Predmet inspekcije: strukture i dizajn programskog rješenja

Izvještaj o pronađenim greškama

Popuniti informacije o pronađenim greškama, te kategorijama u koje spadaju. Lokacija greške u modulu podrazumijeva file i linije koda u kojima se greška nalazi.

Br.	*Check Lista*	*Tip*	*Opis*	*Lokacija*	*Ozbiljnost*
1.	Inspekcija strukture programskog rješenja	U kodu nema funkcija koje se ne pozivaju ni na jednom mjestu	Postoje 2 Funkcije koje se ne koriste	10-ta linija funkcija pod nazivom Swap i 85-ta linija koda funkcija pod nazivom Trade	2
2.	Inspekcija strukture programskog rješenja	U kodu nema ponavljanja koje može biti zamijenjeno jedinstvenom funkcijom	Veliki broj puta se ponavlja razmjena vrijednosti elemenata u nizovima	Razmjene vrijednosti se vrše u sljedećim linijama koda: 31,53,78, 141.169,187	2
3.	Inspekcija varijabli i izraza programskog rješenja	Koristi se jedan stil imenovanja varijabli	Dio varijali je imenovan na Bosanskom a dio na Engleskom jeziku	Varijable na Bosanskom: 69,206,263; na Engleskom 102,129,157,248	2

4.	Inspekcija dokumentacije programskog rješenja	Svi kompleksni dijelovi koda posjeduju komentare	Kompleskni dijelovi klasa ne posjeduju komentare	Klasa AlgoritamCombSort -linija 118; klasa ShellSort linija 51	1
5.	Inspekcija varijabli i izraza programskog rješenja	Nema varijabli koje se ne koriste	Unutar funkcije BubbleSort postoji varijabla velicina koja se ne koristi	Unutar linije 69	1

Izvještaj o metrikama grešaka

Ukupan broj pronađenih grešaka: 5

Normirani broj grešaka: 16

Broj grešaka po LOC: 5/333=0.01501

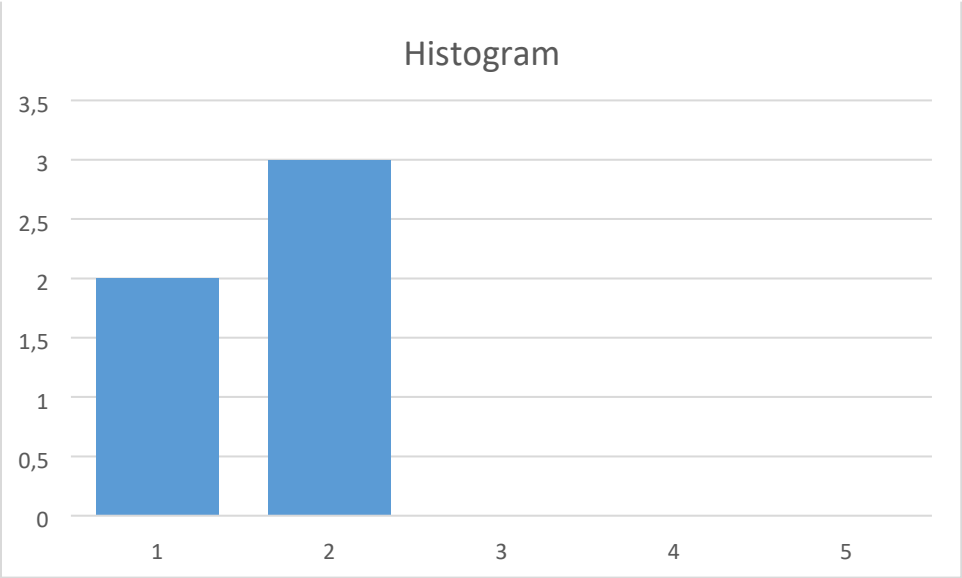
Broj normiranih grešaka po LOC: 16/333=0.04804

Efikasnost otkrivanja grešaka: 2.5 (5 gresaka/2 sata inspekcije)

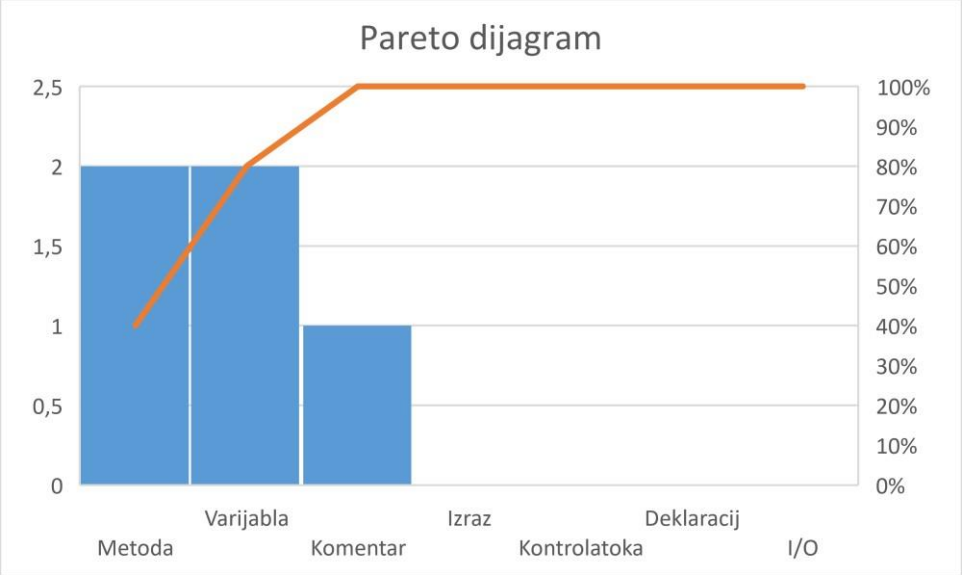
Normirana efikasnost otkrivanja grešaka: 3.0 (5 grešaka/3 sata inspekcije)

Ukoliko je normirani broj grešaka približno jednak broju grešaka bez normiranja dalazimo do zaključka da je ozbiljnost grešaka koje su pronađene u kodu minorne.
Normirana efikasnost otkrivanja grešaka može biti veća od efikasnosti otkrivanja grešaka bez normiranja

Prikaz histograma ozbiljnosti grešaka:



Prikaz Pareto dijagrama za pronađene greške:



2 algoritma koja sam ja implememntira us SelectionSort i ShellSort. Algoritam SelectionSort sam dodijelio kolegi Harunu, a ShellSort kolegi Alenu za review.

```
class AlgoritamSelectionSort
{
    static void Swap(ref int a, ref int b)
    {
        var t = a;
        a = b;
        b = t;
    }
    public static void SelectionSort(ref int[] input)
    {
        for (var i = 0; i < input.Length; i++)
        {
            var min = i;

            for (var j = i + 1; j < input.Length; j++)
            {
                if (input[min] > input[j])
                {
                    min = j;
                }
            }
            if (min != i)
            {
                var temp = input[min];
                input[min] = input[i];
                input[i] = temp;
            }
        }

    }
}
```

```
class AlgoritamShellSort
{
    public static void ShellSort(ref int[] array)
    {
        // Udaljenost između elemenata koji se porede
        var d = array.Length / 2;
        while (d >= 1)
        {
            for (var i = d; i < array.Length; i++)
            {
                var j = i;
                while ((j >= d) && (array[j - d] > array[j]))
                {
                    var temp = array[j - d];
                    array[j - d] = array[j];
                    array[j] = temp;
                    j = j - d;
                }
            }
            d = d / 2;
        }
    }

}
```

Ja sam ze pregled dobio algoritme BubbleSort i CombSort. Od grešaka sam pronašao:

1. Unutar klase AlgoritamCombSort su varijable imenovane na Engleskom jeziku, što se razlikuje od ostatka koda.
2. Unutar klase AlgoritamBubbleSort postoji varijabla velicina i funkcija “Trade” koje se ne koristi.

```
class AlgoritamCombSort
{
    static int getNextGap(int gap)
    {
        gap = (gap * 10) / 13;
        if (gap < 1)
            return 1;
        return gap;
    }

    public static void CombSort(ref int[] input)
    {
        int n = input.Length;
        int gap = n;
        bool swapped = true;

        while (gap != 1 || swapped == true)
        {
            gap = getNextGap(gap);
            swapped = false;

            for (int i = 0; i < n - gap; i++)
            {
                if (input[i] > input[i + gap])
                {
                    int temp = input[i];
                    input[i] = input[i + gap];
                    input[i + gap] = temp;

                    swapped = true;
                }
            }
        }
    }
}
```

```
class AlgoritamBubbleSort
{

    public static void BubbleSort(ref int[] niz)
    {
        int velicina;
        //Opadajući ispis niza
        int temp;
        for (int i = 0; i < niz.Length; i++)
        {
            for (int j = i + 1; j < niz.Length; j++)
            {
                if (niz[i] < niz[j])
                {
                    temp = niz[i];
                    niz[i] = niz[j];
                    niz[j] = temp;
                }
            }
        }
    }
    static void Trade(int[] arr)
    {

        int j = 1;
        int temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
    }

}
```