
Razvoj softvera I

Implementacija softvera



A good programmer is someone who looks both ways before crossing a one-way street.



Doug Linder

Sadržaj



- Implementacija softvera
 - Aspekti efikasnog radno okruženja
 - Programiranje
 - Minimiziranje kompleksnosti
 - Mogućnost promjene/nadogradnje
 - Verifikacijski mehanizmi
 - Primjena standarda
 - Ponovna iskoristivost koda


Implementacija softvera



- Faza implementacije predstavlja jednu od prvih faza u okviru koje se mogu primijetiti nedostaci faze dizajna
 - Dizajn mora biti u potpunosti razumljiv učesnicima ove faze
 - Implementacija bi trebala zadržati sve pozitivne karakteristike dizajna
- Tokom faze implementacije se jednim dijelom opisuju pojedini stavke dizajna koje nisu mogle biti detaljnije razrađene u toj fazi. Gotovo je identično preklapanje i sa fazom testiranja koja započinje već u najranijim fazama implementacije, odnosno programiranja

Implementacija softvera



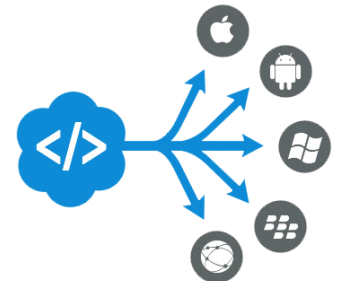
- Najznačajniji akter ove faze je programer 
- Efikasno radno okruženje, često direktno vezano za zadovoljstvo i raspoloženje, podrazumijeva postojanje sljedećih stavki:
 - Hardver i softver
 - Pristup internetu
 - Alat za upravljanje (dijeljenje) programskim kodom
 - Alat za testiranje
 - Profajler
 - Alat za refaktorisanje
 - Kontinuirana obuka i usavršavanje



Hardver i softver



- Pojam softver se u ovoj fazi odnosi na razvojna okruženja koji se koriste za implementaciju softvera, te u zavisnosti od platforme, programskog jezika i konkretnih zahtjeva korisnika govorimo o okruženjima:
 - Visual Studio, Android Studio, Eclipse, RubyMine, PyDev itd.
- Prilikom odabira okruženja potrebno je voditi računa o potrebnim alatima što će svakako odrediti i vrstu licence
 - ...
 - Professional
 - Ultimate
 - ...



Hardver i softver



- Sve pozitivne karakteristike razvojnog okruženja mogu biti eliminisane neadekvatnim hardverskim komponentama
 - Dugotrajno izvršenje često korištenih akcija npr. kompajliranje
- Uštede na hardverskim komponentama predstavljaju adekvatnu poslovnu odluku?
 - Nezadovoljstvo i frustracije programera
 - Efikasnost rada



Pristup internetu

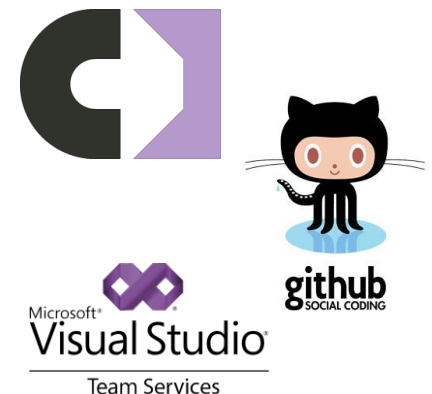


- U zavisnosti od vrste projekata na kojima se radi, programerima se ponekad postavljaju određena ograničenja kada je riječ o pristupu internetu (u okviru radnog vremena). Razlozi za pomenuta ograničenja su različiti:
 - Prevencija napada
 - Zaštita programskog koda
 - Itd.
- Negativni aspekti ovih ograničenja su dosta značajni, posebno sa stanovišta pristupa dokumentaciji, blogovima, forumima i sl.



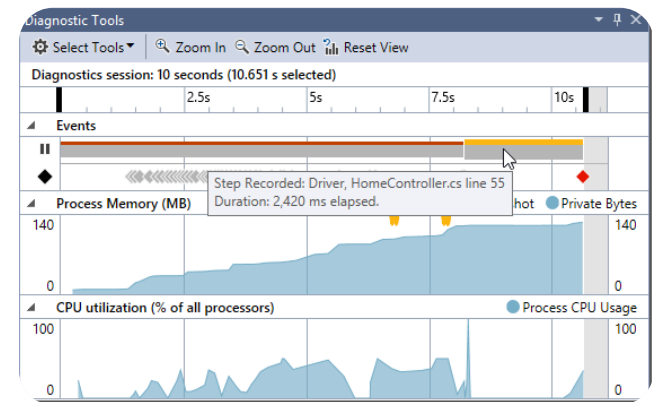
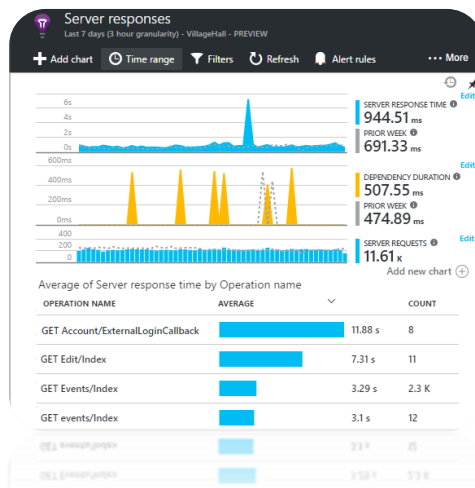
Alat za upravljanje (dijeljenje) programskim kodom

- Pod pojmom alati podrazumijevamo i servise, a oni omogućavaju dijeljenje programskog koda, praćenje promjena na fajlovima i/ili cijelom projektu (historija promjena), te povratak na neku od ranijih verzija
- Neki alati i servisi su namijenjeni dijeljenju samo open-source projekata, dok drugi omogućavaju mnogo značajniju kontrolu nad pristupom i upravljanjem
 - GitHub
 - CodePlex
 - VSTS (Visual Studio Team Services)
 - SVN
 - SourceForge



Profajleri

- Profajleri omogućavaju detaljniji uvid u način izvršenja softvera
 - Alokacija memorije
 - Zauzetost procesora
- Neka razvojna okruženja posjeduju ugrađene alate za profajliranje, dok druga zahtijevaju alate razvijene od strane drugih kompanija





Alati za testiranje i refaktorisanje

- Testiranje se ubraja u jednu od najznačajnijih faza, a pošto je tema narednog predavanja neće biti detaljnije obrađivana u ovoj prezentaciji
- Refaktorisanje - podrazumijeva promjenu koja ne mijenja ponašanje softvera (ne dolazi do promjene funkcionalnosti), ali značajnije poboljšava jednostavnost, fleksibilnost i jasnoću programskog koda



ReSharper



Kontinuirana obuka i usavršavanje

- Organizovanje različitih predavanja, online kurseva ili bilo kojeg drugog oblika edukacije predstavlja jednu od zagarantovano isplativih aktivnosti
- Praksa je pokazala da se efikasnost rada u značajnoj mjeri može povećati ukoliko se posvećuje adekvatna pažnja kontinuiranoj edukaciji



Programiranje



- From design to code
- Veliki broj aplikacija posjeduje operacije koje su mnogo kompleksnije od klasičnog unosa, modifikacije, te prikaza podataka (CRUD). U tom kontekstu je jako bitno odabrati pristup, model ili algoritme koji će osigurati najadekvatnije rješenje:
 - Sortiranje podataka
 - Brzu pretragu zapisa unutar baze
 - Pronalaženje najoptimalnije rute na mapi
 - Enkripciju i dekripciju podataka
 - Itd.
- Gotovo svi algoritmi posjeduju određene prednosti i nedostatke
- Prilikom modifikacije algoritma voditi računa o narušavanju efikasnosti

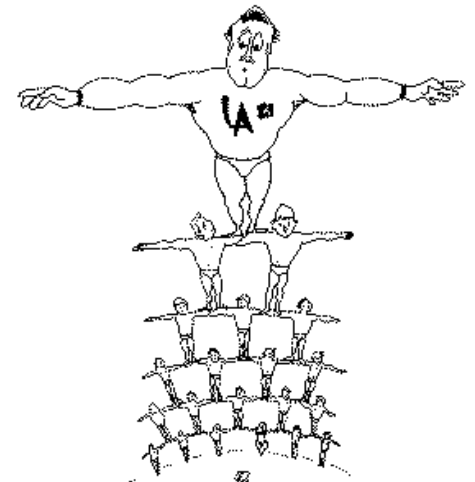


Programiranje



- From design to code. How?
- Zadatak: Generisati raspored nastave
- Pristup: Top-Down dizajn
 - Podijeliti problem na manje dijelove
 - Nastavi dijeliti sve dok svi koraci ne budu potpuno jasni
 - U potpunosti razumjeti svaki korak

```
GenerisiRaspored() { }
```





Programiranje

```
GenerisiRaspored() {
```

```
1. za svaki predmet u nastavnom planu i programu
```

```
  A. preuzmi broj sati predavanja i vježbi
```

```
  B. preuzmi zahtjeve za opremom
```

```
  C. preuzmi nastavno osoblje
```

```
    I. za svakog člana nastavnog osoblja na predmetu
```

```
      $. preuzmi preference nastavnog osoblja
```

```
      $. za svaku učionicu koja ispunjava zahtjeve
```

```
        #. provjeri zauzetost učionice za preferirani dan nastavnika
```

```
          *. dodijeli slobodan termin posmatranom predmetu i članu  
             nastavnog osoblja
```

```
      $. ukoliko nije pronađen termin za posmatranog člana nastavnog  
         osoblja
```

```
        #. ...
```

```
}
```

Implementacija softvera



- U okviru faze implementacije softvera je potrebno voditi računa o sljedećim aspektima:
 - minimiziranju kompleksnosti
 - budućim promjenama/nadogradnji
 - verifikacijskim mehanizmima
 - primjeni standarda
 - ponovnoj iskoristivosti koda



Implementacija softvera



- *Minimiziranje* ili potpuno eliminisanje *kompleksnosti* predstavlja veoma značajan aspekt u svim fazama razvoja, a posebno u fazi implementacije.
- Učesnici razvoja trebaju preferirati čistiji, pregledniji i jednostavniji kod u odnosu na kompleksan i dojmljiv kod
 - složeniji programski kod obično zahtijeva mnogo više vremena za pisanje i testiranje
 - složeniji kod je teže debugirati, održavati i nadograđivati
 - novi učesnici na projektu veoma često imaju poteškoća da shvate, te ukoliko je potrebno nadograde kompleksnije dijelove koda

Implementacija softvera



- Značajan napredak u minimiziranju kompleksnosti se može postići:
 - primjenom standarda
 - modularnim dizajnom
 - primjenom adekvatnih tehnika programiranja
- Pomenuto je u direktnoj vezi sa *budućom nadogradnjom*
 - Danas je gotovo nemoguće pronaći softver koji od svoje prve verzije nije doživio niti jednu promjenu (nadogradnju, modifikaciju)
 - Za očekivati je da će i na trenutnoj verziji softvera biti neophodne nadogradnje



Implementacija softvera



- Implementirani dijelovi softvera trebaju biti *što jednostavniji za verifikaciju*, bilo da je vrše programeri, tester i ili čak krajnji korisnici
- Osiguranje verifikacijskih mehanizama zahtijeva pružanje podrške za analizu koda i jedinično testiranje (engl. Unit testing)
- Adekvatna verifikacija zahtijeva izbjegavanje kompleksnih i teško razumljivih programskih struktura i algoritama



Implementacija softvera



- Ponovna iskoristivost koda u značajnoj mjeri može utjecati na produktivnost, kvalitet i troškove razvoja
- Ponovna iskoristivost se može posmatrati sa dva aspekta:
 - konstrukcija za ponovnu iskoristivost – tokom razvoja pojedinih dijelova softvera vodi se računa o načinu organizacije koda i ponovnoj iskoristivosti
 - konstrukcija sa ponovnom iskoristivošću – korištenje postojećih dijelova koda prilikom konstruisanja novog softvera



Implementacija softvera



- *Primjena* internih i eksternih *standarda* trebalo bi da osigura pozitivne efekte u svim aspektima implementacije
- Standardima u fazi implementacije se obično definišu:
 - komunikacijske metode – standardi za korištene formate dokumenata i sadržaja
 - programski jezici – standardi korištenih programskih jezika
 - standardi programiranja (kodiranja) – imenovanje klasa, objekata, varijabli, metoda i sl.
 - alati – standardi korišteni prilikom modeliranja kao naprimjer UML



Implementacija softvera



Avoid using abbreviations unless the full name is excessive.
Avoid abbreviations longer than 5 characters.
Any Abbreviations must be widely known and accepted.
Use uppercase for two-letter abbreviations, and Pascal Case for longer abbreviations.
Do not use C# reserved words as names.
Avoid naming conflicts with existing .NET Framework namespaces, or types.
Avoid adding redundant or meaningless prefixes and suffixes to identifiers

Example:

```
// Bad!  
public enum ColorsEnum {...}  
public class CVehicle {...}  
public struct RectangleStruct {...}
```

Do not include the parent class name within a property name.

Example: `Customer.Name` NOT `Customer.CustomerName`

Example:

```
// Bad!  
Void WriteEvent(string message)  
{...}  
  
// Good!  
private Void WriteEvent(string message)  
{...}
```

Do not use the default ("1.0.*") versioning scheme. Increment the `AssemblyVersionAttribute` value manually.

AssemblyVersionAttribute value manually
Do not use the default ("1.0.*") versioning scheme. Increment the

Always use the built-in C# data type aliases, not the .NET common type system (CTS).

Example:

```
short NOT System.Int16  
int NOT System.Int32  
long NOT System.Int64  
string NOT System.String
```

Only declare member variables as `private`. Use properties to provide access to them with `public`, `protected` or `internal` access modifiers.

private, protected or internal access modifiers
Only declare member variables as private. Use properties to provide access to them with

Implementacija softvera



- Standardi koji se mogu primijeniti u kontekstu programiranja (kodiranja) uključuju sljedeće:
 - tehnike za kreiranje čitljivog programskog koda (imenovanje instanci, formatiranje koda, i sl.)
 - organizaciju programskog koda (imenski prostori, biblioteke, rutine)
 - korištenje kontrolnih struktura
 - obrada izuzetaka
 - prevencija sigurnosnih propusta na nivou koda (buffer overflow, pristup elementima van opsega niza/kolekcije, i sl.)
 - dokumentovanje programskog koda
 - korištenje resursa uz pomoć adekvatnih mehanizama (tredovi/niti, zaključavanje korištenog reda na nivou baze, i sl.)

Implementacija softvera



- Interna (unutrašnja) dokumentacija sadrži informacije namijenjene onima koji će imati uvida u programski kod projekta
- Uobičajeno je da se na početak svake komponente stavljaju osnovne informacije o komponenti, a one obično uključuju sljedeće:
 - naziv komponente
 - autora komponente
 - opis uloge komponente u cjelokupnom dizajnu sistema
 - vrijeme pisanja i eventualne revizije programskog koda
 - korištenje podataka, algoritama i drugih resursa neophodnih za rad komponente

Implementacija softvera



- Eksterna (spoljna) dokumentacija je namijenjena onima koji možda nikada neće imati uvid u programski kod projekta
 - omogućava malo opširnije pojašnjenje nego što je to moguće ostvariti sa internom dokumentacijom (komentarima u programu)
 - sadrži opšti pregled komponenti sistema
 - od objektno-orientisanih komponenti, opšti pregled objekata i klasa trebalo bi da opiše glavne interakcije između njih
 - dizajn predstavlja okosnicu spoljnje dokumentacije koja se dopunjava tekstualnim opisom pojedinosti koda komponente
 - korisnicima ili programerima daje uvid u tok podataka na nivou komponente

KRAJ PREZENTACIJE

