Pravila izrade softvera *

* software craftsmanship

Dejan Tomić 1026/2014
Razvoj softvera 2, 2014/15

9 osnovnih ideja

- Pobedi složenost
- Izaberi svoj proces
- Piši programe prvo za ljude
- Programiraj u svoj jezik
- Fokusiraj se uz pomoć konvencija
- Programiraj u pogledu domena problema
- Pazi na odron kamenja
- Stalne iteracije, iznova i iznova
- Razdvoj softver i religiju

Pobediti složenost (1)

- ❖ Glavni cilj razvoja softvera
- ❖ Gomila principa za upravljanje složenošću

- Podeliti sistem na podsisteme
- Pažljivo definisati interfejse
  - I čuvati njihovu apstraktnost
- Izbegavati globalne podatke
- Izbegavati duboke hijerarhije nasleđivanja
- Izbegavati mnogo nivoa petlji i uslova
- Izbegavati “goto”
- Pažljivo odrediti svoj pristup obradi grešaka
- Biti sistematičan sa ugrađenim sistemom izuzetaka

Pobediti složenost (2)

- Ne puštati klase da previše porastu
- Održavati rutine kratkim
- Koristiti jasna imena promenljivih (“šta je i a šta j?”)
- Smanjiti što više broj parametara rutine
- Pratiti konvencije

- Konvencije – proizvoljne
- Apstrakcija, najveća dostignuća
  - Prelaz na više programske jezike (Fred Brooks 1995)
  - Rutine
  - Klase i paketi
- Imenovanje promenljivih: “šta” a ne “kako”
- OOP – apstrakcija nad algoritmima i podacima

Izabrati svoj proces

- Izabrani proces razvoja mnogo znači
- Pri timskom radu: organizacija više znači od iskustva pojedinaca
- Primer: posledice nestabilnih zahteva
  - Proces određuje i stabilnost zahteva i potreban stepen stabilnosti
  - Za manje stabilne zahteve: inkrementalni razvoj
  - Greške u zahtevima su mnogo skuplje od grešaka u razvoju
- Prerana optimizacija je pogrešna
  - Velike izmene se prave na početku, male na kraju
- Procesi niskog nivoa: pisanje pseudokoda
  - “Top-down” dizajn; postojanje komentara u kodu
- “Kod je bitan, a ne neki apstraktan proces”?
  - Nije dobro razmišljanje
  - Dobar proces iskorišćava mozak na najbolji način

Pisati programe za ljude (1)

- **Prvo za ljude, pa za računare**
- Pažnja posvećena čitljivosti koda
  - Računarima nije bitna čitljivost koda; bitna je drugim ljudima
- Čitljivost pozitivno utiče na:
  - Obimnost
  - Količinu grešaka
  - Debagovanje
  - Mogućnost izmene koda
  - Vreme razvoja (posledica svega navedenog iznad)
  - Kvaliteta proizvoda (posledica svega navedenog iznad)
- Lakše je proveriti da kod radi ako je čitljiv
- Bolje je pisanje dobrog koda nego čitanje lošeg
- “Šta ako pišem kod sam za sebe?”
- Douglas Comer (1981): privatni i javni programi
  - Posebna pažnja pri pretvaranju “privatnog” programa u “javni”

Pisati programe za ljude (2)

- U stvarnosti, velike su šanse da će neko drugi morati da menja vaš kod
  - Thomas 1984: na održavanju prosečnog programa radi 10 generacija programera
  - Ti programeri potroše 50-60% vremena razumevajući tuđi kod
  - Parikh & Zvegintzov 1983

Programiraj u svoj jezik

- A ne *u njemu*
- Ne treba se ograničavati na ono što jezik podržava
- Isto tako, ne treba koristiti nešto samo zato što to jezik podržava (primer: globalni podaci, goto)
- Smisli prvo šta želiš da postigneš, pa odluči kako
  - Programski jezik ne podržava assert()? Napiši svoj assert()!
  - Ne podržava enumeracije? Implementiraj ih!
- Ponekad alat može da bude dovoljno primitivan
  - Pa je potrebno menjati ceo pristup programiranju
  - Najčešće će razlika između željenih ciljeva i onoga što alat podržava samo tražiti malo slaganja sa okruženjem

Koristiti konvencije

- Dosta detalja je proizvoljno
  - Uvlačenje koda; formatiranje komentara; redosled rutina klase...
  - Nije bitan konkretan način, već raditi na isti način svuda
  - Izbegava donošenje istih odluka iznova
- Konvencije ukratko iznose bitne informacije
  - Jedno slovo može da razlikuje lokalne i klasne promenljive
  - Prvo malo/veliko slovo može da razlikuje tipove, konstante i promenljive
  - Uvlačenje koda, poravnanje...
- Mogu da posluže izbegavanju problema
  - npr nekorišćenje globalnih promenljivih, korišćenje zagrada oko izraza, postavljanje pokazivača na NULL...
- Daju predvidljivost zadacima
- Mogu da nadoknade slabosti jezika
- Ne treba imati ni previše, ni premalo konvencija

Programirati u domenu problema

- Raditi na najvećem nivou apstrakcije
  - Raditi na nivou samog problema, a ne rešenja
- Nije bitno kako je nešto urađeno
  - Komentari tipa “i je indeks reda u tom i tom fajlu” nisu potrebni
  - Ime promenljive treba da izbegne detaljan komentar o svrsi
- Deljenje programa u više nivoa apstrakcije
  - Nivo 0: mašinske instrukcije i funkcije OS-a (za niže jezike)
  - Nivo 1: alati i strukture programskog jezika
  - Nivo 2: implementacione strukture (stek, red, stablo, algoritmi...)
  - Nivo 3: rad sa problemom, na nižem nivou (delovi problema)
  - Nivo 4: rad sa problemom, na višem nivou (razumljiv svima)
- Neke tehnike nižeg nivoa
  - Klase, interfejsi
  - Imenovane konstante kao značenje “literala”
  - Promenljive međurezultata
  - Buleanske funkcije kao razjašnjenje kompleksnih testova

Znaci upozorenja

- Programiranje nije potpuno umetnost, niti nauka. To je “zanat” negde između ta dva. (McConnell 2004)
- Obratiti pažnju na upozorenja – mogućnost problema
- “Ovo je lukav kod” – znak lošeg koda
- Klasa sa većim od prosečnog broja grešaka
- Dobar proces treba da spreči nastajanje grešaka
  - Do trenutka testiranja, većina grešaka bi već bila otklonjena
  - Previše debagovanja – ne radi se “pametno”
- Korišćenje metrika (daju meru kvaliteta)
- Svako upozorenje – sumnja u kvalitet programa
- Rad na ponovljenom kodu – sumnja u organizaciju
- Da li je kod težak za razumevanje?
  - Ne bi trebalo “shvatati” kod – već ga čitati

Znaci upozorenja (2)

- Pravite sopstvena upozorenja
  - Myers, 1978 o (ne)otkrivanju grešaka
  - Napravite program tako da se greške teško mogu “propustiti”
- Upozorenja kompajlera!

Iterativni razvoj

- Tokom specifikacije sistema – više verzija zahteva
- Isporuka sistema u više navrata (inkrementalno)
- Prototipovi, više alternativnih rešenja – tip iteracije
- Najvažnije su iteracije nad zahtevima projekta
- Dizajn softvera je heuristički proces
  - Softver teži da bude proveren – iterativno testiranje i razvoj
- Optimizacija softvera – u iteracijama
  - Ne postoji neka opšta tehnika optimizacije – često se dešava da kod bude još sporiji nakon pokušaja optimizacije
- Ocenjivanje koda (review) – provera kvaliteta
  - Dok ne prođe proveru, ponavljaju se iteracije
- Fred Brooks, 1995: “Build one iteration to throw away; you will, anyhow”
- Inženjerstvo: uraditi za 20c što drugi mogu za 1$

Razdvojiti softver i religiju

- Poštovanje jednog metoda dizajna, određenog načina formatiranja, pisanja komentara...
- Metode prvo treba isprobati pre nego što se proglase dobrim ili lošim
  - “Snake oil” – ne treba verovati da je neka metoda zaista dobra zato što neko tako kaže (“ovo će rešiti sve vaše probleme”)
- Potrebna je selektivnost radi nalaženja najbolje metode razvoja