Проблематике софтверског заната

Стојановић Стеван 1082/2015

Математички факултет
Универзитет у београду

Развој Софтвера 2

Побеђивање сложености

- Подела система у мање делове.
- Пажљиво дефинисање класних интерфејса.
- Чување апстракције класних интерфејса.
- Избегавање коришћења података са глобалним опсегом.
- Избегавање сложених хијерархија.
- Избегавање сложених петљи.
- Избегавање коришћења наредби скока.

Побеђивање сложености

- Пажљив приступ руковања грешакама.
- Систематична употреба уграђених механизама за руковање изузецима.
- Не дозвољавати класе монструозне величине.
- Држати се кратких субрутина.
- Коришћење разумљивих имена за променљиве.
- Минимизација параметара субрутина.
- Придржавање конвенција.

Бирање процеса

- На великим пројектима организација пројекта има много већи значај него индивидуалне способности програмера у пројекту.
- Процес који се користи утиче на саму стабилност захтева као и на то колико стабилни треба да буду захтеви
- За постизање флексибилности може да се користи инкрементални приступ развоја софтвера
- Квалитет мора да буде уграђен од почетка
- Прерано оптимизовање је једна грешака.
- Мишљење да је код све што је битно је кратковидно и игнорише брда експерименталних и практичних доказа о супротном

Писање кода за људе

- Читљивост има позитиван утицај на цео пројекат :
  - Разумљивост.
  - Прегледност.
  - Количину грешака
  - Дебаговање
  - Лакоћу мењања кода
  - Време проведено правећи код.
  - Квалитет кода.
- Боље је писати читљив код него више пута читати нечитљиви код.
- чак и ако код пишете само за себе боље је писати читљив код , ради касније употребљивости.

Програмирање у језику

- Немојте да се ограничавате само концептима који су аутоматски подржани језиком који користите
- Не морате да користите наредбе скока или податке са глобалним опсегом само зато што су подржани.
- Програмирање коришћењем најочигледније путање није обавезно и најбоља идеја.

Конвенције

- Конвенције чувају прогамера да немора да прави исте произвољне одлуке непрестано
- Конвенције побољшавају читљивост и синхронизацију кода на пројектима где ради већи број програмера.
- Конвенције могу да садрже корисне информације.
- Конвенције штите од познатих опасности
- Када има превише конвенција , постаје напорно памтити их све.

Програмирање у домену проблема

- Владање сложеношћу тако да се ради на највећем могућем нивоу апстракције.
- Код високог нивоа би требало да описује проблем који се решава , и да буде напуњен дескриптивним именима класа и позивима субрутина које тачно објашњавају шта програм ради
- Код на вишем нивоу не би требало да брине о томе како су чувани подаци.
- Подела програма у нивое апстракције :
  - Ниво 0 Операције оперативног система и машинске инструкције.
  - Ниво 1 Структуре и алати програмског језика.
  - Ниво 2 Имплементационе структуре ниског нивоа.
  - Ниво 3 Ниски ниво израза домена проблема.
  - Ниво 4 Високи ниво израза домена проблема.

Програмирање у домену проблема

- Технике ниског нивоа за рад у домену проблема :
  - Коришћење класа за имплементирање структура које имају смисла у домену проблема.
  - Скривање информација о типовима података ниског нивоа и детаљима њиховог имплементирања.
  - Коришћење именованих константи за обележавање смисла речн и нумеричких литерала.
  - додељивање непосредних резултата променљивима ради њиховог обележавања.
  - коришћење логичких функција за разјашњавање комлексних тестова.

Знаци пажње

- "Компликован код"могућ знак лошег кода.
- Добар процес неће дозволити да се прави код који је склон грешкама.
- Много дебаговања на пројекту је упозорење да људи на пројекту можда не раде паметно.
- Метрике дизајна такође дају неку идеју о квалитету кода.
- Када вам је јасан концепт дизајна , имплементациони детаљи нижег нивоа не би требало да праве проблем.
- Ако је код тешко разумљив , погрешно је написан , учините код једноставнијим.
- Програмирајте тако да сами правите упозорења.
- Истраживање корекције дефекта је нашла да је најчешћи разлог пропуштање грешака.Грешке су биле видљиве излазима тестова , али нису биле примећене.

Итерација (Понављање)

- Понављање је погодно за више активности развоја софтвера , током првих спецификација система треба радити са корисником кроз више верзија захтева док нисте сигурни да се слажете са њим.
- Итерација на захтевима је можда важно колико и било који други концепт развоја софтвера
- Развој софтвера је хеуристички процес , и као такав је подложан ревизијама и побољшањима.
- Када је софтвер функционалан можете преписати неке делове кода да га побољшате у целини.
- Када је итерација доведена до екстрема долази до неефикасности.
- Трик софтверског инжињерства је направити одбациву верзију што пре и што јефтиније могуће.

Развој софтвера и религија

- Религија се у развоју софтвера јавља у много форми , у догматичном поверењу у један метод дизајна , у ватреном одбијању рада са глобалним променљивима итд.
- Софтверски пророци ће покушати да вам продају нову узбудљиву методу која је лек за све болести.
- Користите више метода , слободно експериментишите са новим и занивљивим методама , али испробајте и старе поузданије методе.
- Електицизам је користан став при дизајну софтвера , ако методе сматрамо алатима у кутији алате , понакад није битно који алат изаберете за посао , али понекад прави значајну разлику који алат изаберемо.
- Важно је пажљиво бирати алат.

Експериментисање у развоју софтвера

- Треба експериментисати кроз цео процес развоја софтвера.
- Многи од религиозних приступа развоја софтфера настају због страха од прављења грешака.
- Развој софтвера је прављење малих грешака ради избегавања прављења великих грешака.
- На сваком нивоу где сте спремни да направите избор , вероватно можете да направите експеримент чији ће резултат бити приступ који је најпогоднији за дати избор.
- Поента је да треба држати отворен ум у свим аспектима развоја софтвера.