TNM094 – Medietekniskt kandidatprojekt

Implementation

Konvertera design till programkod

- Detta är icke-trivialt
  - allt är inte löst med arkitektur och design
  - diagram, tabeller och strukturer måste omvandlas till kod
  - koden måste vara lätt att läsa och förstå
  - dra nytta av styrkan hos både språk och design

- Kodningsstandard
  - konventioner för namn, mellanrum, radbryt, kommentarer, etc.
  - organisations-specifik standard som underlättar delad kod
    (e.g. Google style guide, K&R, KNF, Gnu style, etc.)

[Pfleeger]

Generella programmerings-tips

- Gör koden så generell som är praktiskt
- Isolera input och output
- Säkerställ att förutsättningar är uppfyllda
  - använd assert(...)
  - kasta exception
- Revidera din kod
  - Kontrollera läsbarhet
  - Kontrollera implementation
  - Skriv om på ett bättre sätt

[Pfleeger]

Kontrollstrukturer och algoritmer

- Keep-It-Simple-Stupid (KISS)
  - Det finns ingen prestige i att skriva svårläst kod
  - Offra *inte* läsbarhet och struktur för prestanda
- Läsbarhet – uppifrån och ned
  - Beslut → handling undvik att bryta ett naturligt flöde
  - Undvik långa eller djupa block (if, for, while, …)
  - Samla kod som hör ihop
  - Göm detaljer i makron och subrutiner
- Variabel- och komponent-namn
  - Använd längre, beskrivande namn vid större räckvidd

[Pfleeger]

Par-programmering

- Grund-princip
  - två likvärdiga programmerare
  - en förare/pilot skriver programkod
  - en navigator granskar och ger återkoppling i realtid
  - rollerna byts ut med jämna mellanrum
- Effekt
  - närmre samarbete mellan programmerare
  - kontinuerlig utvärdering av designval
  - att förklara leder till att man får bättre förståelse
  - ökad chans att man hittar potentiella problem

[Pfleeger]

Par-programmering

- Viktigt att känna till
  - två mindre erfarna programmerare vinner mer än erfarna
  - produktivitet och kvalitet ökar inte med tiden
  - att skriva kod tar mer resurser i anspråk
  - koden blir marginellt bättre strukturerad och har färre fel
  - blandade resultat presenteras i litteraturen

[Pfleeger]

Kod-dokumentation

- Extern kod-dokumentation
  - system-perspektiv
  - övergripande arkitektur och programdesign
  - beskrivande diskussion kring hur koden är strukturerad
- I Agil utveckling
  - läggs ofta i en ”README”-fil

[Pfleeger]

Kod-dokumentation

- Intern kod-dokumentation
  - Kommentarer längst upp i filen
  - Kommentarer kopplade till klass och funktion
  - Ytterligare kommentarer
- Varför dokumenteras inte kod?

  - Arrogans

    'Min kod är så tydlig och elegant att dokumentation inte behövs'
  - Uppskjutande

    'Det är mycket att göra just nu – jag gör det senare'
  - Obskurantism

    'Om ingen annan förstår min kod så kan de inte sparka mig'
- Viktiga verktyg
  - Automatisk kod-analys och dokumentationsgenerator
  - Doxygen, Javadoc, etc.

[Pfleeger]

Kommentarer längst upp i filen

- Information om själva filen
  - komponentens namn, syfte och plats i arkitekturen
  - författare
  - revisionshistorik
  - eventuellt även diskussion kring innehållet
    och använda datastrukturer, algoritmer och kontroll

[Pfleeger]

Klass-/funktionsdokumentation

- Placeras i koden som kommentar
- Information om klassen/funktionen
  - förutsättningar, parametrar och resultat
  - typiskt användande, relaterade klasser och funktioner
  - algoritmer som används, pseudokod
  - kvalitetsaspekter – trådsäkerhet, stabilitet, tidsåtgång, etc
- Kan skrivas före implementationen
  - lättare att beskriva utifrån anroparens perspektiv
  - Documentation Driven Dev. (jmf TDD, MDD, etc.)

[Pfleeger]

Kommentera mera

- Information om själva programkoden
  - tydlig kod *reducerar* behovet av kommentarer
  - kommentarer kostar inget vid körning
  - få med tankar, instruktioner och pseudokod
- Vissa standarder
  - Gör det lättare att hitta viktiga anteckningar
  - TODO – markerar ofärdig implementation
  - HACK – markerar att programmet kringgår en bugg

[Pfleeger]