TDDI02

Programmeringsprojekt. Föreläsning 3

Jonas Lindgren, Institutionen för Datavetenskap, LiU

På denna föreläsning:

- Verifikation, Validering och Testning
- XP – Extreme Programming

Vad är ett fel?

I engelskan kan man skilja på 3 olika typer..

- Error

  Ett misstag begånget av en person under utvecklingen, som kan resultera i..

- Fault

  Ett fel i programvaran, som när det exekveras kan orsaka..

- Failure

  Något oönskat händer under exekvering!

Obs: Errors/faults måste inte automatiskt leda till failures, inte nödvändigtvis en 1-1 mappning mellan de tre åt något håll.

Validering/Verifikation

..är *nästan* testning, det med, fast på ”dokument”!

Kravanalys:

- Kompletthet – Allt är med, inget underförstått
- Konsistens – Inga motsägelser
- Genomförbarhet – Inte bara rent funktionellt, men även ur ett affärsmässigt perspektiv
- Testbarhet – Kommer det gå att testa kraven, avgöra om de är uppfyllda eller ej?
  - Skapa testdata för funktionella tester?

Validering/Verifikation (cont.)

Design:

- Finns konsistens mellan kravspecifikation och design, beskriver de samma sak?
- Håller designen rätt kvalitet? (Företagsstandarder, etc..)
  - Skapa testdata för funktionella tester?

Implementation:

- Finns konsistens mellan implementationen och designen, beskriver de samma sak?
- Testa implementationen, uppfyller den de tidigare satta kraven?

Validering/Verifikation (cont.)

Testfasen:

- Testa moduler (unit tests)
- Integrationstesta modulerna som samarbetar
- Systemtest
  - Alphatest? Betatest?
- Acceptanstest

Underhåll:

- Alla steg i mindre skala, om och om igen

Statisk testning

- Läs/granska
  - Låt andra göra, själv har man inte rätt ”destruktiva” inställning
- Review – som ovan fast mer anonymt och ”institutionaliserat”
- Walkthrough/Inspection
  - Gå igenom med några inspektörer, resulterar i ett åtgärdsprotokoll
  - Inspection den mer formella av de två, resultaten rapporteras vidare
  - Formella bevis
  - Översätt kod till logiknotation
  - Bevisa från startvillkor att slutvillkoren kommer ske
- Statisk program-analys

Walkthrough/Inspection lista

- Felaktig användning av data

  Oinitierade variabler, arrayindex utanför gränser, ”dangling pointers”
- Deklarationsfel

  Användning av icke-deklarerade storheter, dubbel deklaration i ett block
- Beräkningsfel

  Division-by-zero, overflow, typmismatchningar, operatorprioriteter
- Logikfel

  < istället för <=, and-or, operator-prioriteter
- Fel i styrning

  Oändliga loopar, varvräknarfel
- Interfacefel

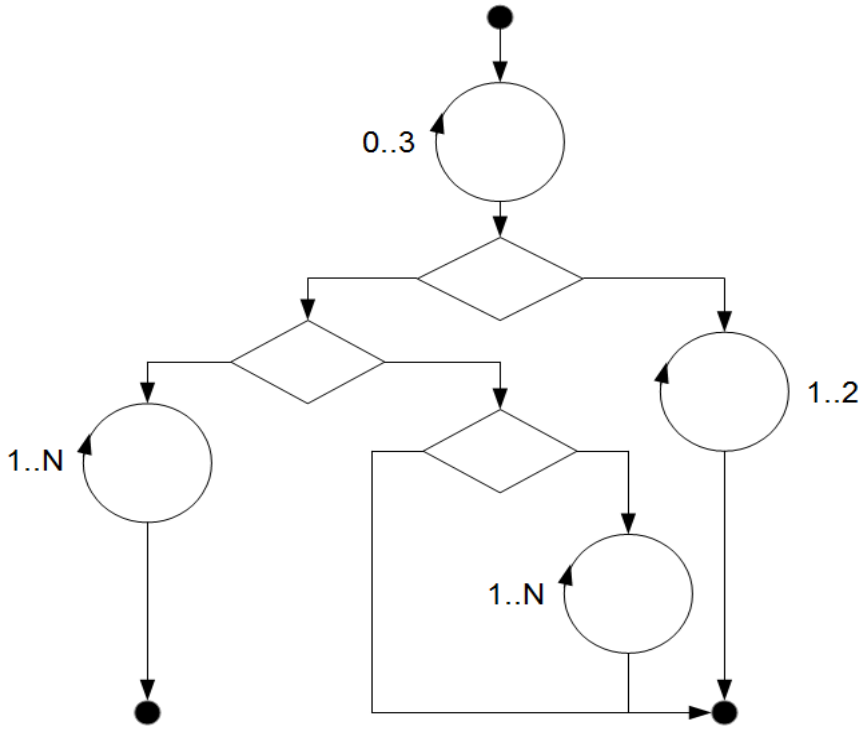
  Antalet, typerna på parametrar, globala data

Dynamisk testning

- Funktionell – Black box testing
  - Titta ej på koden!
  - Testar bara funktionalitet utifrån specifikationen
- Strukturell – White box testing
  - Titta på koden!
  - Testa de ”svåra delarna”
    - Logik i villkorssatser, if, for, while, etc..
    - Extremvärden
  - Egentligen vill man täcka all kod, få hög ”coverage”
- Hitta lämplig testdata
  - Identifiera och testa ”gränsövergångar”!
- ”Unit testing”

Coverage testing

Hur många sätt går det att ta sig från början till slut?



4 x ((N + (1 + N)) + 2) ???

”Unit testing”

” **A unit test is an automated piece of code that invokes a *unit of work* in the system *and then checks a single assumption about the behavior of that unit of work*.**”

Ett Unit-test är:

- Automatiserbart
- Isolerat
- Konsistent
- Läsbart
- Lättunderhållet
- Pålitligt

```
// can it add the numbers 1 and 1?
public void testSumOneAndOne() {
   Adder adder = new AdderImpl();
   assert(adder.add(1, 1) == 2);
}
```

C++ unit test ramverk:
Google Test, QtTest, unit++, etc..

Regressionstestning

- Ändringar kan introducera nya buggar
- ..eller *återintroducera* gamla buggar
- Använd tidigare skrivna tester, se om de fortfarande passerar
  - Unit tests?

Software System

A system made up of software, hardware, and data that provides its primary value by the execution of the software.

- Good Software System meets requirements
- Good Software System has appropriate architecture
- Good Software System is maintainable, extensible and testable
- Good Software System has low support cost

IVAR JACOBSON INTERNATIONAL

Testning – exempel

”Specifikation”:

Ett program läser in tre heltalsvärden från en rad. De tre värdena tolkas som längderna på de tre sidorna i en triangel. Programmet skriver ut ett meddelande som anger om triangeln är likbent, liksidig eller om alla tre sidor är olika.

Uppgift:

Skriv ett antal testfall (dvs ”kategorier” plus några specifika testdata per kategori) som du anser testar programmet på ett bra sätt.

Testning – exempel (cont.)

```
#include <iostream>
using namespace std;

int main() {
   int x, y, z;
   cin >> x >> y >> z;
   cout << x << ” ” << y << ” ” << z << ”\n”;
   if (x == y && y == z) {
      cout << ”Triangeln är liksidig.\n”;
   } else if (x == y || x == z || y == z) {
      cout << ”Triangeln är likbent.\n”;
   } else {
      cout << ”Alla sidor i triangeln är olika.\n”;    }
   return 0;
}
```

Testning – exempel (cont.)

```
Unix > make testprogram
…
Unix> testprogram
1 1 100
Triangeln är likbent.
Unix> testprogram
0 0 0
Triangeln är liksidig.
Unix> testprogram
-1 -1 -1
Triangeln är liksidig.
Unix> testprogram
3.0 3.0 3.0
Alla sidor i triangeln är olika.
Unix> testprogram
Klas, Kalle och Lotta
Triangeln är liksidig.
```

Testning – exempel (cont.)

Självevaluering av testfall:

1. Har du testfall som representerar en giltig oliksidig triangel? Obs, fall som 1 2 3 eller 2 5 10 gör inte det, de är inte giltiga trianglar!
2. Har du testfall som representerar en giltig liksidig triangel?
3. Har du testfall som representerar en giltig likbent triangel? Obs, ett testfall som har 2 2 4 är inte en giltig triangel!
4. Har du åtminstone tre testfall som representerar giltiga likbenta trianglar, så att du provar alla tre permutationer av två lika sidor (t.ex. 3 3 4, 3 4 3 och 4 3 3)?
5. Har du ett testfall där en sida är 0?
6. Har du ett testfall där en sida är negativ?

Testning – exempel (cont.)

7. Har du ett testfall med tre heltal större än noll där summan av två är lika med ett tredje? Med andra ord, om programmet anser att t.ex. 1 2 3 representerar en oliksidig triangel, så är det en bug (då det ej är en giltig triangel)!
8. Har du åtminstone tre testfall i kategori 7 så att du provar alla permutationer där en sida är lika med summan av de övriga två (t.ex. 1 2 3, 1 3 2, 3 1 2, etc..)?
9. Har du ett testfall med tre heltal större än noll så att summan av två är mindre än det tredje (t.ex. 1 2 4 eller 12 15 30)?
10. Har du testat minst tre permutationer av ovanstående?
11. Har du ett testfall där alla sidor är 0?
12. Har du minst ett testfall med värden som inte är heltal?

Testning – exempel (cont.)

13. Har du åtminstone ett testfall som specificerar fel antal värden?
14. *För varje testfall ovan, specificerade du (i förväg!) det förväntade resultatet?*

Testning – minnespunkter

- Utarbeta och dokumentera tester, inklusive förväntat utfall, i förväg!
- Gör alltså inga tester ”i flykten”!
- Vid buggar, skriv ett test för att påvisa buggen..och sedan bekräfta att den är borttagen!
- Ett test kan aldrig bevisa frånvaron av fel – bara påvisa förekomsten!
- Ett test som inte påvisar något fel är förmodligen ett undermåligt test..eller?
- När ska man sluta testa?

XP – Extreme Programming

Vad är extremt med XP?

- Moment genomförs kontinuerligt
- Utvecklingscykler är små och korta
- Lite dokumentation utanför koden skrivs

Vad krävs/önskas?

- Inte för stora projekt
- Kundrepresentant på plats
- Planerings-”spel”, ”user stories”
- Gärna ”parprogrammering”

XP – Extreme Progra.. (cont.)

Karakteristiskt för XP:

- Utgår från utvecklares och kunders behov
- Ingen stor administrativ apparat, inte jättemycket dokumentation
- Smidig hantering av förändringar
- Snabba resultat, god kvalitet förväntas
- Ingen egentlig designspecifikation
- Kanske inte heller sedvanlig kravspecifikation!

Planering:

- Täta möten mellan kund – utvecklare (~2 veckor?)
- Planerings-”spel” – ”user stories”
- Små, täta uppdateringar (releases) till kunden
- Kund deltar i utvecklingsprocessen

XP – Extreme Progra.. (cont.)

Design:

- Ingen specifikation i förväg
- Enhetstester och testfall produceras före motsvarande kod, är en form av specifikation för enheten
- Ofta automatiserat testande
- Acceptanstest görs för varje ”story”
- Refactoring – Våga bygga om
  - Byt namn på metod/variabel/etc
  - Gör en bit kod till egen metod/motsvarande
  - Flytta på metoder/etc
  - Verktyg finns
  - Görs för ökad läsbarhet och minskad komplexitet

XP – Extreme Progra.. (cont.)

Kodning:

- Parprogrammering
- Kollektivt ägande
- Kontinuerlig integration
- ”Fysisk närhet” mellan utvecklare

Dokumentation:

- Vissa kundkrav
- Utöver det kräver XP inte mycket
- Koden är det viktigaste dokumentet, ska vara av hög kvalitet
- ”Story”na, diagram, etc..
- Acceptanstest kan fungera som kravspecifikation