Projektnamn

Introduktion till testing av it-system HT 2014

|  |  |
| --- | --- |
| Adam Alfa | adal1234 |
| Beatrice Beta | bebe5678 |
| Caesar Gamma | caga9012 |
| Diana Delta | dide3456 |

# Introduktion

En kort introduction till projektet där ni också listar de verktyg ni använt. Om ert versionshanteringssystem går att komma åt ska adressen dit finnas med, annars ska det finnas en länk från vilken man kan ladda hem källkoden.

# Slutlig design

En övergripande modell över systemet. Lämpligt format är ett eller flera klassdiagram, plus eventuella andra modeller som behövs för att förstå hur systemet är uppbyggt. Diagrammen ska vara läsbara. Det är dock fullständigt okej att de är detaljerade, bara det går att zooma in ordentligt på dem. Ett tips är att börja med ett översiktligt diagram som inte innehåller mer än paket och klassnamn, och att sedan lägga till mer detaljerade diagram efter det.

# Testdriven utveckling – process

En översikt över hur ni tillämpat TDD med exempel från olika personer och olika faser i projektet. Om ni har använt versionshanteringssystemet ordentligt bör all information som efterfrågas här finnas i det. Tänk på att kodexemplen ska vara läsbara.

# Testdriven utveckling – erfarenheter

En diskussion om vilka era erfarenheter ni dragit av att tillämpa TDD. Det finns inget rätt eller fel här. Enda sättet att bli underkända är att bara fuska över punkten och säga något pliktskyldigt.

# Ekvivalensklassuppdelning – namn på del

En kort presentation av vad ni valt ut för att tillämpa ekvivalensklassuppdelning på. Ni ska kort motivera valet, och ge tillräckligt med information för att det ska gå att bedöma er. Detta avsnitt och de tre föjande (till och med testmatrisen) ska finnas för samtliga delar ni tillämpat ekvivalensklassuppdelning på.

# Ekvivalensklasser – namn på del

Samtliga ekvivalensklasser för denna del presenterade på ett tydligt sätt.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ekvivalensklassindelning av indata för Konstruktorn för klass ”Discount”.** | | | | |
| Variabel | Valida ekvivalensklasser | Invalida ekvivalensklasser | Gränsfall | Anteckningar |
| minimum | (1) >0 | (2) <0 | (3) =0 | Rabatt ges oavsätt köpt mängd om minimum=0 |
| value[%] | (4) >0 && <100 | (5) <0  (6) >100 | (7) =0  (8) =100 |  |
| value[abs] | (9) >0 && <y | (10) <0  (11) >y | (12) =y  (13) =0 | ”y” är priset på varan innan någon rabatt. |

# Testfall – namn på del

Testfallen som ni fått fram från ekvivalensklasserna. Observera att vi inte vill ha någon kod här, utan bara en tydlig presentation av testfallen i någon lämplig tabellform.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | minimum | value[%] | value[abs] | Förväntat resultat |
| 1 | 8 | 4 |  | Ok |
| 2 | 5 |  | 3<y | Ok |
| 3 | -1 | 4 |  | fail |
| 4 | 8 | -1 |  | fail |
| 5 | 8 |  | -1 | fail |
| 6 | 8 | 120 |  | Fail |
| 7 | 8 |  | >y | fail |
| 8 | 0 | 4 |  | ok |
| 9 | 8 | 0 |  | fail |
| 10 | 8 | 100 |  | fail |
| 11 | 8 |  | =y | fail |
| 12 | 8 |  | =0 | fail |

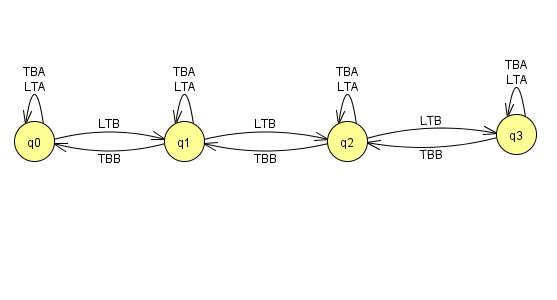
# Testmatris – namn på del

En testmatris som visar sambandet mellan ekvivalensklasserna och testfallen för denna del.

x-led = testfall, y-led = ekvivalensklassindelning

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 1 | x |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  | x |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  | x |  |  |  |  |
| 4 | x |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  | x |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  | x |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  | x |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  | x |  |  |
| 9 |  | x |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  | x |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  | x |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |  |  |  | x |  |
| 13 |  |  |  |  |  |  |  |  |  |  |  | x |

# Tillståndsbaserad testning

En kort presentation av vad ni valt ut för att tillämpa tillståndsbaserad testning på och vilket täckningskriterium ni valt att använda er av. Ni ska kort motivera valen, och ge tillräckligt med information för att det ska gå att bedöma er. Glöm inte att ta med själva modellen.

Vi har skapat en scenario där varor placeras till och från ett kvitto. Tillstånden baseras på antalet bananer som finns i kvittot, 0 till 3 (q0 till q3). Som täckningskriterie har vi valt tillståndstäckning (state coverage) och från det har vi skapat ett testfall. Vi upptäckte snabbt att ett testfall enkelt täckte hela scenariot så vi körde på det.

LTA = Lägg till annan (Produkt)

TBA = Ta bort annan (Produkt)

LTB = Lägg till banan

TBB = Ta bort banan

**Scenario**

Först börjar Kalle med tomt kvitto q0. Kalle köper ett kaffe paket (LTA), sedan köper han en banan (LTB) och hamnar på q1, Kalle ångarar sig och lämnar tillbaka kaffepaketet (TBA) och då är han fortfarande på q1.

Kalle köper en tomat (LTA) och då står han fortfarande på q1, Kalle köper en till banan (LTB) och då hamnar hamnar han på q2, Kalle köper en melon (LTA) och är fortfarande på q2, Kalle ångrar sig och lämnar tillbaka en banan (TBB) och hamnar han på q1, Kalle köper en appelsin (LTA) och är han fortfarande på q1, Kalle köper en till banan (LTB) och då hamnar han på q2, Kalle ångrar sig och lämnar tillbaka en banan (TBB) och hamnar då på q1, Kalle ångrar sig och lämnar tillbaka en banan (TBB) och hamnar på q0, Kalle köper en banan (LTB) och hamnar på q1, Kalle köper en till banan (LTB) och hamnar på q2, Kalle köper en till banan (LTB) och hamnar på q3 och då är alla tillstånd är täckta.

# Testfall för tillståndsbaserad testning

Testfallen som ni fått fram från tillståndsmaskinen. Observera att vi inte vill ha någon kod här, utan bara en tydlig presentation av testfallen i någon lämplig tabellform. Det ska enkelt gå att mappa testfallen till tillståndsmaskinen.

|  |  |  |
| --- | --- | --- |
| ID | Beskrivning | Täckta tillstånd |
| 1 | 1. Påbörja inhandlandet av varor 2. Köp kaffe 3. Köp banan 4. Ångra kaffe 5. Köp tomat 6. Köp banan 7. Köp melon 8. Ångra banan 9. Köp apelsin 10. Köp banan 11. Ångra banan 12. Ångra banan 13. Köp banan 14. Köp banan 15. Köp banan | 1. q0 2. q0 3. q1 4. q1 5. q1 6. q2 7. q2 8. q1 9. q1 10. q2 11. q1 12. q0 13. q1 14. q2 15. q3 |

# Granskning

En kort presentation av den del av koden ni valt ut för att göra en formell granskning av och processen ni använt er av inklusive eventuella checklistor, scenarier, edyl. Ni ska kort motivera valen, och ge tillräckligt med information för att det ska gå att bedöma er.

# Granskningsrapport

En lista över de påträffade felen och hur pass allvarliga ni bedömer dem.

Punkt 7: metoden getLineSubTotal i classen Receipt är otydlig, exempel på alternativ: getItemSubtotal

Metoden getMinimum i klassen Discount är otydlig, exempel på alternativ: getDiscountMinimum, getMinimumPurchaseAmount.

getValue : getDiscountValue

punkt 8: inga metoder har någon koll på variabel värden, en lösning är att använda assert satser.

Punkt 9: inga metoder har någon koll på returvärden, en lösning är att använda assert.

Punkt 13:

Variabel namnet ”p” för en Product-objekt i metoden add i klassen Receipt är inte beskrivande.

Variabel namnet ”lineIndex” för i metoden getItemSubTotal i klassen Receipt är inte beskrivande.

Metoden getTotal i klassen Receipt har variabel namn som ”e” och ”p”, dessa är inte beskrivande.

Instance variabel namnet ”minimum” är inte beskrivande.

value : discountValue

type : typeFlag

punkt 43 till 51:

kommentarer existerar ej.

# Granskning – erfarenheter

En diskussion om vilka era erfarenheter ni dragit av att tillämpa granskning. Det finns inget rätt eller fel här. Enda sättet att bli underkända är att bara fuska över punkten och bara säga något pliktskyldigt. Ni förväntas förhålla er till såväl kursboken som utdelat material och IEEE Std 1028.

# Kodkritiksystem

*En presentation av de problem som hittats med hjälp av verktyg för statisk analys och en diskussion av dem enligt anvisningarna. Det räcker alltså inte med att bara lista problemen, ni måste förhålla er till dem också. Tänk också på att ni ska göra detta både på koden som den såg ut före granskningen och på koden efter att ni rättat det som kommit fram under granskningen.*

Vi använder oss av **FindBugs** som kodkritiksystem. Systemet rapporterade inga fel.

# Statiska mått

En presentation och diskussion kring ett antal lämpliga statiska mått på koden. Att vi inte specificerar exakt vilka mått som ska tas upp beror på att olika verktyg har olika uppsättningar, men vi förväntar oss fler och mer intressanta mått än bara rena storleksmått som LOC, #klasser, #metoder, etc. Även här är det viktigt att förhållas sig till måtten, inte bara lista dem.

# Täckningsgrad

En översikt över vilken täckningsgrad era testfall uppnått. Denna kan antagligen tas rakt av från verktyget ni använt för att mäta den. Om ni inte uppnått fullständig täckning så ska detta förklaras och motiveras.

Vi valde statement coverage som täckningsgrad. Vi har uppnått 100% täckning. Vi använde oss av ÄckelEmma för att se våran coverage.



# Profiler

En kort presentation av hur ni gått tillväga för att testa koden med en profiler och vilka resultat ni fick fram.

Våran projekt är inte så lämplig att profilera eftersom programmet körs och avslutas omedelbart; den sitter inte och väntar eller arbetar på något. Vi profilerade programmet genom att skapa ett litet main() metod där vi skapar några produkter & rabatter och lägger dom i ett kvitto. För att profilera programmet lägger vi en brytpunkt vid sista kod satsen, kör programmet i debug mode och låter den sitta där medan vi utforskar resultaten från profilern. Vi använder oss av profilern ”JVM Monitor”.

Det finns inte så mycket intressant att titta på; vi har en tråd, som i vårt fall är pausad och använder i princip ingen CPU, det mest intressanta är väll minnet. Här är några highligts: Högst upp på listan har vi *char[]* som använder 456KB minne, vi har 1521 stycken *Class* pekare som använder 183KB minne, *String* pekare har vi 6575 stycken av och de använder 157KB minne.

Vi använder oss av *double* till mycket: värdet för pengar, värdet för kvantitet av produkt osv. Vi använder oss av 4 st *double[]* och det tar upp 336 bytes!

# Byggscript

Byggscriptets första (seriösa) version, och den slutliga.

Här följer våran första fungerande byggscript. Den kommer förmodligen att vara den slutliga.

<?xml version="1.0" encoding="UTF-8"?>

<project name="InteProj" default="runjunit">

<target name="runjunit" depends="compile" description="Run Forest! Run!">

<junit printsummary="on">

<test name="inteproj.AllTests"></test>

<classpath>

<pathelement location="./resources/junit.jar"></pathelement>

<pathelement location="./resources/hamcrest.jar"/>

<pathelement location="./compiled"></pathelement>

</classpath>

<formatter type="plain" usefile="false"/>

<formatter type="plain"/>

</junit>

</target>

<target name="compile" description="Do the macarena!">

<javac srcdir="./src;./test\_src" destdir="./compiled" includeantruntime="false"

classpath="./resources/junit.jar"></javac>

</target>

</project>

# Övrigt

Här kan ni ta upp övrigt av relevans för bedömningen av ert arbete. Om avsnittet inte behövs kan det plockas bort.