Kvittoprojektet

Introduktion till testing av it-system 2015-10-25

|  |  |
| --- | --- |
| Jacob Ståhl | Jast8350 |
| Albin Fagernes | Alfa9590 |
| Filip Gezelius | Fige7684 |
| Josef Rosen | Joro8913 |

Obs! Ändra inte på formatet, det ska vara liggande A5.

# Introduktion

En kort introduktion 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.

Vi satt tillsammans och spånade vilket projekt vi ville utföra. Vi kom till den slutsatsen att vi ville bygga kvittohanteringsystemet. Vi kände att de låg oss närmast i vår kunskapskapacitet och att vi kunde ta oss an den utmaningen tillsammans. Efter att vi valt ett projekt så diskuterade vi vilka verktyg vi ville använda oss utav. Vi insåg att de vore mest praktiskt att använda oss utav de verktyg som finns att tillgå i skolan och de vi har lärt oss på föreläsningarna. Så valet blev att vi programmerar med Java i Eclipse och använder oss utav Junit för testsyftet. Versionshantering har vi arbetat med Github för lagringen och med klienten TortoiseGit för själva hanteringen. Se länken nedan för att få åtkomst till vårt repositorium.

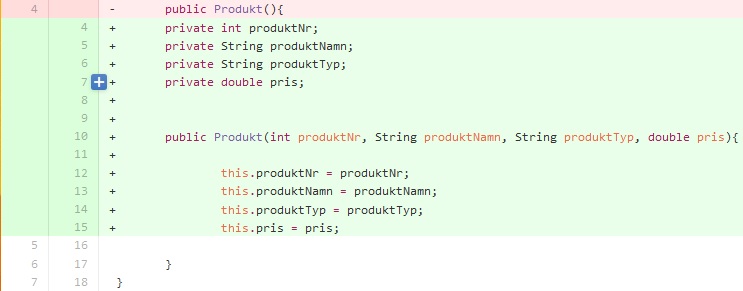
<https://github.com/JacobStahl/TestProject>

Efter att vi bestämt oss för alla verktyg började vi spåna fram en klasstruktur, dvs. hur vi skulle bygga våra klasser och testklasser. Vi gjorde även en uppdelning av arbetsuppgifter, dvs. vem som ska göra vad.

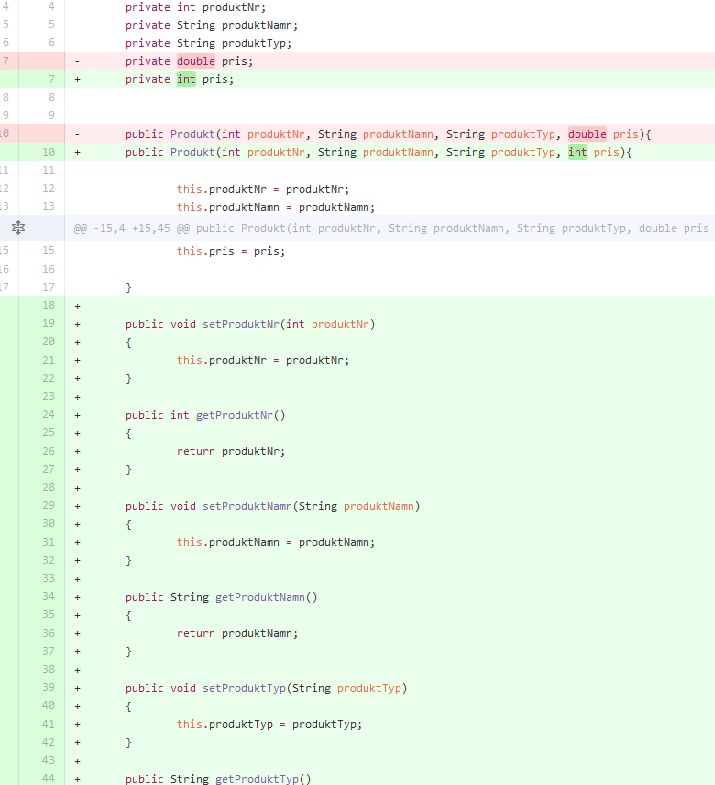
# Slutlig design

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

Vi har tillämpat testdriven utveckling till i stort sätt 100 % av de vi har utvecklat. Vi har hela tiden byggt testen innan vi har implementerat metoderna som testerna använder sig av. När vi har testat systemet med vårt teckningsverktyg uppgår det till ca 99 % täckning, vilket är en bra siffra. Vi har valt att inte testa vissa saker då det är för simpla för att göra tester på. Nedan visar vi några kodexempel på hur vi har arbetat enligt TDD. Dessa exempel är hämtade från vårt projekt på GitHub.  
  
Först har vi skapat en produktklass med tillhörande konsturktor för att definiera en början på en klass utan metoder.  


Sedan har vi konstruerat testerna i enlighet med vårt projekt.  


Efter att vi har byggt alla tester så bygger vi sedan metoderna i produktklassen.  


# Testdriven utveckling – erfarenheter

Genom hela kursen har vi fokuserat på den teoretiska biten om testdriven utveckling. Detta projekt gav oss möjligheten att faktiskt tillämpa de teoretiska kunskaperna praktiskt. Till en början var det svårt att göra testerna innan man byggde metoderna. Det kändes som att man gjorde emot allt man gjort innan. Så det tog ett tag innan man kom igång med TDD och att de kändes naturligt att arbeta med.

När vi väl hade kommit igång med ett testdrivet arbetssätt så började allt flyta på och vi byggde alltid testerna innan vi konstruerade metoderna. Fördelen vi har upplevt med TDD är givetvis att vi har mindre fel i koden än vad man kanske hade innan när man utvecklade. Nackdelen vi kände i början var att vi tyckte de tog lång tid att skriva en klass eftersom testerna tog så mycket tid innan man kunde påbörja programmerandet av klassen. Men eftersom man förhoppningsvis tjänar in denna tid i slutänden så känns de värt det.

En annan fördel vi känner är att man inte behöver debugga lika mycket som vanligt. I och med att testet utgör en mall för metoden så blir metoden sällan fel. Det känns även som att man bygger enklare och effektivare kod eftersom man endast gör det minsta möjliga för att få testet att fungera. En fördel är också om man ändrar någonstans i koden så kan man köra alla tester för att se om någon annan programsats har påverkats av den ändringen.

Förhoppningsvis får vi tillfälle att arbeta testdrivet när vi sedan efter utbildningen kommer ut på arbetsmarknaden. Det känns som denna kurs har vart väldigt lärorik och tagit oss till en ny nivå programmeringsmässigt.

# 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.

# 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.

# Testmatris – namn på del

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

# 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.

# 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.

# 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.

# 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.

# 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.

# Profiler

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

# Byggscript

Byggscriptets första (seriösa) version, och den slutliga.

# Övrigt

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