LINKÖPINGS UNIVERSITET

TDP005 Projekt: Objektorienterat system

Reflektionsdokument

Författare

Alexander Jonsson, alejo720@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2016
Version 1.1

2016-12-17

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
1.1	Korrekturläsning	2016-12-17
1.0	Första version, påbörjad renskrivning från kladd	2016-12-16

2 Föregående reflektionsdokument

I föregående reflektionsdokumentet skrev jag att jag inför nästa projekt ska se till att båda använder Git från första början och det lyckades vi med under detta projekt. Nästan det första vi gjorde var att skapa en Git repository till projektet.

I förra reflektionsdokument skrev jag även att inför nästa projekt skulle jag inte använda mig av ShareLaTeX. Detta berodde på att jag var tvungen att vid varje ändring ladda ner filerna från ShareLaTeX för att sedan ladda upp dem på GitLab. Detta leder till att det blir svårt att jobba för flera på samma dokument. Därför använde vi oss inte av ShareLaTeX utan installerade istället IDE:n TeXstudio vilket finns gratis att installera på Linux och Windows. Detta ledde till att båda samtidigt kunde jobba med dokumentet genom git istället.

Dessa förbättringar visar hur viktigt det kan vara att reflektera sitt projekt för att det ska gå bättre på nästa projekt.

3 Vad har jag fått ut av kursen, vad har jag lärt mig?

Vad har jag fått ut och lärt mig av kursens alla moment? Vilka fördelar och nackdelar finns det med de olika arbetssätten?

3.1 Kravspecifikation samt agilt arbete

Under kursen hade vi en beställargrupp som skulle agera beställare av vårat spel. Vanligtvis fås en kravspecifikation av en beställare men i detta fall skapade vi vår egen kravspecifikation. Även fast vi tillsammans har diskuterat fram en spelidé skulle vi kunna ha olika perspektiv på spelidén. Därför är det bra att få ner spelidén konkret i en kravspecifikation. I en kravspecifikation finns det både ska och bör-krav och tanken är sedan att tillsammans med beställaren flytta upp bör-krav till ska-krav då det är lämpligt. Detta skapar ett agilt arbetssätt vilket ger fördelen att arbeta med ett krav i taget och lägga till nya krav. Det ger även möjligheten till att ha uppdelade deadlines istället för bara en deadline då alla krav (hela spelet) ska vara färdiga. Detta utövades i praktiken genoma att vi hade möte med beställargruppen varje vecka där vi flyttade upp bör-krav till ska-krav.

Trots att vi inte hade en riktig beställargrupp som gav oss kravspecifikation tycker jag ändå kursen har gett mig ökad förståelse i hur det är att jobba mot en kund agilt.

3.2 Designspecifikation och klassdiagram

Under kursen skapade vi även en designspecifikation vars tanke är att förklara hur vårat spel skulle vara designat, allt ifrån klassuppbyggnad till mappstruktur. Detta inkluderade klassdiagram vilket jag tycker är en viktig del inför programmering av ett objektorionterad program. För det första leder det till en bra struktur innan utförandet av projektet. Samtidigt blir det lättare för alla i projektet om det finns en fast mall att jobba ifrån. Med bra designspecifikation blir det också lättare för någon som inte varit med och skrivit designspecifikationen att förstå designen på spelet. T.ex. krävs det förståelse av de logiska klassernas uppbyggnad för att någon ska implementera grafiken som baserar sig på de logiska klasserna. Det är samma princip som med kravspecifikationen: Det hjälper att få ner strukturen konkret på papper så att alla har samma tolkning på upplägget av projektet. För det andra, så kan fundering och diskussion av designen leda till att komma på saker i förväg, saker som kanske skulle ledit till att behöva strukturera/skriva om sitt program vilket kan bli tidskrävande.

Jag och Joakim spenderade stor del på vårat klassdiagram vilket jag för det mesta tyckte var bra. Vi tänkte

verkligen till hur klasserna skulle vara uppbyggda för att fungera så bra som möjligt, och varje gång vi jobbade med klassdiagrammet kom vi på nya saker vi inte tänkt på innan. Detta ledde till att implementationen av klasserna flöt på nästan utan några problem alls. Däremot, när vi väl satte igång och kodade märkte vi fortfarande några ändringar som krävdes på våra klassdiagram. Jag tror att det är i princip omöjligt att skapa ett perfekt klassdiagram innan kodandet. Därför ska jag nästa gång vid skapadandet av klassdiagram inte sitta överdrivet länge med klassdiagrammet.

Sammanfattningsvis, leder dessa positiva fördelar med klassdiagram till att jag ska försöka så ofta som möjligt skapa klassdiagram inför objektorienterad programmering i framtiden.

3.3 Testdriven utveckling

Vi provade testdriven utveckling under kursen. Detta utfördes genom att efter att ha skapat ett klassdiagram skapa olika testscenarion för klasserna för att sedan steg för steg implementera delar av klasserna. Jag fann flera fördelar samt nackdelar med testdriven utveckling. En fördel är att vid skrivning av testerna tvingades klasserna att abstraheras så mycket som möjligt, vilket är själva grundtanken med att skapa en klass. Dessutom testas automatiskt gamla testscenarion hela tiden vid ny implementation eller ändrning, vilket ser till att allt fortfarande fungerar som det ska. Däremot finns det risk för att tänka fel och skapa felaktiga tester. Detta kan leda till svårigheter att hitta felen då första tanken som dyker upp inte är att det skulle vara fel i testet. Detta hände några gånger för oss vilket ledde det till att vi efter ett tag började kolla på om testfallet var fel innan vi tittade om implementationen var fel, vilket förstör hela poängen med att skriva testerna.

En annan nackdel är att om det nu skulle vara så att en relativt stor del av programmet måste skrivas om kan det krävas att skriva om flera tester vilket kan vara tidskrävande.

Om vi skulle haft en kund skulle det vara svårt att jobba på vårat sätt. Vi implemterade all logik genom testdriven utveckling innan grafiken. Detta leder till att det inte skulle bli lika tydligt för kunden hur långt vi hade kommit och vilka bör-krav som ska flyttas upp till ska-krav. Trots dessa nackdelar tycker jag att testdriven utveckling i det stora hela är ett bra arbetssätt som i längden sparar tid.

3.4 Doxygen och dokumentation

Under kursen har jag lärt att kommentera enligt doxygen vilket var ett väldigt smidigt verktyg för att generera dokumentation till kod utan att ha med koden. Dokumentation är viktig för att både sig själv och andra ska förstå vad delar av koden verkligen har för funktion. Eftersom det är bra att ha så lite koppling mellan klasser som möjligt kommer klassen troligtvis vilja användas i andra tillfälllen än till det nyvarande projektet, och kanske till och med av någon annan programmere. Då kan det vara väldigt smidigt att ha en doxygen-genererad dokumentation utan kod att kolla i. Därför kommer jag i framtida programmering försöka få vanan att alltid skriva min dokumentation i doxygen format, ifall jag skulle vilja generera en doxygen dokumentation.

3.5 Granskning och förståelse av någon annans kod

Under kursen fick vi granska någon annans kod vilket gjorde oss tvungna att sätta sig in i någon annans kod. Detta kan vara väldigt svårt men är något som ofta behövs göras i ett projekt. Därför tycker jag att jag fick ut mycket av detta moment. Dessutom fick jag också lära mig hur andra tänker och kodar.

4 Samarbetet

I stort sett tycker jag samarbetet i gruppen har fungerat bra. Med liten debatt om spelidén, gemensamt intresse och då båda gillar att diskutera fram lösningar flöt projeket på. Då vi konstaterade tidigt att jag

föredrar att hålla på med logiken av ett program och Joakim grafiken delades arbetet mellan varandra natuligt upp i det vi föredrog. Dock så jobbade vi för det mesta nästan lika mycket med båda delar eftersom det oftast blir lättare att komma på lösningar på problem vid diskussion.

5 Annat

Något jag tyckte var svårt var att skapa generella klasser. Oftast krävs det extra tillägg i klassen för att den specifikt ska fungera i just sitt program. Ett exempel är att en spelare kanske inte alltid vill kunna hoppa och ducka men i vårat fall var vi tvungna att implementera det. Vi hade en spelare och en projektil som båda var en "entity". Då borde t.ex. funktionen som kollar om en spelare kolliderar med en projektil vara generell och ta in en entity instället för specifikt en projektil som den gör nu. Detta tyckte vi dock var okej eftersom kollideringen av en projektil specifikt beror på projektiltillståndet jämfört med spelarens tillstånd.