TDDC76 Projekt - Samlade erfarenheter

Student 1

I början av projektet hade vi många möten för att diskutera igenom varje dokument som skulle skapas. Detta var kanske för noggrant gjort men samtidigt känner jag, som inte är så van programmerare (jämfört med andra i gruppen), att jag fick större förståelse för projektet som helhet. I OOD:n definierade vi alla funktioner som behövdes i alla klasser redan innan vi började programmera, vilket var bra när man skulle förstå någon annans kod. Behövde man veta något mer än vad som var kommenterat stod det oftast i OOD:n.

Vi valde att arbeta med en klass var och implementera den. Detta uppfattade jag som riktigt bra, eftersom man då kunde sitta på var sitt håll och arbeta. Det förenklades också av att vi hanterade allt via en SVN. Vi kunde då jobba parallellt och ha alla kommentarer samlade på ett ställe, så vi kunde se hur långt de andra kommit samt deras kommentarer.

Vi kommenterade mycket under kodningen, vilket var bra. Min erfarenhet är att det är bättre att kommentera för mycket än för lite. Det hjälpte mycket när man gjorde ändringar eller när funktioner behövde omdefinieras och det blev enklare att gå in i någon annans kod när man behövde ändra.

Något att tänka på är att allt tar mycket längre tid än man har tänkt sig, vilket vi märkte mot slutet. Man vill inte gärna jobba dubbelt i slutfasen för att få saker att fungera i tid.

Student 2

Den kanske främsta erfarenheter från projektet är hur nyttigt det var att göra OOA och OOD innan gruppen satte igång med själva programmerandet. Arbetet gick då lätt att dela upp och hela gruppen behövde inte sitta tillsammans och programmera. Jag personligen hade stor nytta av OOD-dokumentet när jag utformade min del. Utan den hade jag haft många frågor om vilka funktioner som fanns att tillgå i de klasser som min del skulle kommunicera med.

Förutom OOD:n hade jag stort utbyte av de lite kortare möten som gruppen höll under hela programmeringsfasen. Vi kunde då stämma av att allas arbetade fortskred som tänkt och vi kunde ställa frågor till övriga gruppmedlemmar.

Något som hade varit bra att ha gjort i samband med programmeringsfasens inledning hade varit att göra en tidsplan. Vi gjorde en tidsplan först efter någon vecka och inte förrän då började projektet ta fart. Något annat som hade varit bra att tänka på är att felsökning kan ta lång tid, så det är viktigt att lägga upp en tidsplan med ganska mycket utrymme på slutet.

Rent programmeringsmässigt har jag framförallt lärt mig nyttan av att skriva snygg kod och kommentera. Saknades kommentarer var det svårt att förstå hur en funktion som man själv inte skrivit fungerade. Dessutom har jag förstått hur man kan använda undantag på ett bra sätt. Jag har dessutom lärt mig mycket kring programmeringen, som till exempel att använda versionshanterare som SVN eller hjälpprogram som Doxygen, program som jag inte använt innan projektet.

Student 3

Ett lyckat programmeringsprojekt kräver en hel del planering och även om det var mycket möten innan vi faktiskt fick något gjort, så lönade det sig senare. Mycket tid lades på att få alla på samma spår i projektet och det var bra att vi hade ett par vana programmerare som kunde styra upp strukturen på koden och komma med idéer om hur man kunde förenkla vissa algoritmer.

Under arbetet tog vi hjälp av flera verktyg, som SVN, Doxygen och WebCollab, för att hålla reda på vad som var gjort. Även om en del behövde sättas upp manuellt för att använda dem var det väl värt mödan.

Tid gick även åt för att sätta upp utvecklingsmiljöer åt alla, men med hjälp av egna datorer och fjärrskrivbord fick alla tillgång till en välutrustad utvecklingsmiljö i Linux, vilket gjorde att alla kunde arbeta i samma miljö.

Vad jag som dokumentansvarig missade var att från början använda LaTex till de dokument som skulle produceras. Detta hade förenklat dokumentprocessen betydligt, eftersom SVN då hade kunnat versionshantera dokumenten fullt ut.

Uppdelningen i projektet var bra, genom att alla fick tilldelat arbete efter förmåga och där var och en kunde utvecklas. Projektet delades upp så att var och en fick en klass som primärt ansvarsområde, vilket gjorde att mycket av koden kunde utvecklas parallellt. Till detta fick alla tilldelat en av de andra klasserna, som man skulle hjälpa till med och som den primärt ansvariga kunde bolla idéer med. Det var en god tanke men i praktiken behövdes det inte i ett så litet projekt som detta.

En sak att tänka på är att inte krångla till saker för mycket, men samtidigt inte vara rädd för att testa nya saker i språket. Till exempel genom att använda *templates* kunde vi förenkla mycket av databehandlingen. Samtidigt hade vi ett par fall av diskussioner i gruppen om användningen av datatyper och strukturer bara för att de fanns, vilket enbart hade försvårat arbetet.

En viktig programmeringslärdom som framkommit var att vi lite sent upptäckte hur data i std::map är ordnade. Vi trodde att man skulle få ut data i samma ordning som man stoppade in dem, vilket visade sig vara fel. Lösningen var dock enkel, då en std::vector med data lagrade i par leder till exakt samma syntax som för map då man hämtar ut data från vectorn.

Vid programmering av GUI, eller egentligen med vilket bibliotek som helst, är ofta den främsta informationskällan den dokumentation som finns på Internet, vilken ofta kan vara bristfällig. Något man då kan komma ihåg är att det som regel finns ett IRC-forum där man kan fråga och där man ofta får snabb och bra hjälp.

Man bör också ta sig tid att förstå strukturen i de bibliotek man arbetar med. Många fel kan undvikas genom att man funderar på hur olika klasser, funktioner och variabler egentligen hänger ihop och vad de har för inverkan. Exempel, som ofta finns, har en viktig roll i förståelsen men det är viktigt att man inte bara kopierar dessa rakt av.

Det avslutande arbetet med att sätta ihop klasserna till ett fungerande program hade kunnat skötas bättre. Som det blev satt ett par personer och försökte få kompileringen att gå igenom, och eftersom några inte hade testkompilerat koden innan fanns det en hel del triviala fel, som det tog tid att fixa. Just i den fasen är det dock svårt att jobba många samtidigt och det var inte arbetsviljan som var problemet. Med mer tid hade arbetet kunnat delats upp bättre och felkontrollerna hade kunnat göras grundligare.

Student 4

Mina erfarenheter av programmeringsprojektet är framför allt hur viktigt det är att vara noggrann i designfasen, eftersom detta underlättar mycket i implementeringsfasen. Är man noggrann i början blir alla i projektet på det klara med vad som ska göras och det blir lättare att utföra programmeringsuppgiften. Ett bra designarbete gör också att man kan dela upp arbetet i moduler som var och en sedan arbetar med, eftersom man i designfasen bestämmer gränssnittet till dessa moduler.

En annan viktig erfarenhet är att det måste vara möjligt att arbeta delvis självständigt i gruppen om man ska hinna med allt. Det är inte möjligt att alla sitter och programmerar på samma sak, då detta är ett stort resursslöseri. Detta gör också att alla inte behöver lägga ner tid på att lära sig allt om alla delar. För att det självständiga arbetet ska fungera bör man ha avstämningsmöten med regelbundna intervall, till exempel en eller två gånger i veckan. Dessa möten behöver inte vara speciellt långa. Vid sidan av det självständiga arbetet är det också bra att ibland sitta i grupp och programmera. Även om man sitter med var sin modul kan man då snabbt ställa frågor vid behov, både om de andras moduler och om programmeringen i allmänhet.

Student 5

Jag tyckte att det var bra att lägga mycket tid på analys- och designfaserna. Detta gjorde att vi visste vad vi skulle göra och hur, men kanske ännu viktigare var att vi hann upptäcka flera saker där vi trodde olika om vad vi hade bestämt tidigare.

Jag tycker att vi lyckades dela upp arbetet bra. Man fick ansvar för en klass eller några närliggande och hade lite koll på de klasser man skulle samarbeta med, men behövde inte sätta sig in så mycket i hur de andras delar fungerade. Detta gjorde att vi kunde programmera självständigt och effektivt. Eftersom jag var ansvarig för att rita upp graferna var jag inte speciellt beroende av hur långt resten av projektet kommit och kunde testa koden allteftersom, vilket var smidigt. Nu efteråt tycker jag att jag kanske skulle läst in mig mer på hur grafritarprogrammet PLplot fungerade och i mindre mån härmat exemplen.

Det var bra att ha deadline men vi hade behövt bestämma dem tidigare, inte efter en veckas programmerande. Jag som var någon slags projektledare skulle också haft bättre koll på att de efterlevdes. Vi satte upp tider när olika delar skulle vara klara men jag kollade inte med alla att det faktiskt följdes (och det gjordes det inte). Eftersom deadline inte hölls fick vi ont om tid när programmet väl skulle sättas ihop. Tidsbristen gjorde att några fick göra det mesta arbetet med att sätta ihop programdelarna och få dem att fungera tillsammans, eftersom det inte fanns tid för övriga att sätta sig in i de andras kod och hur det hela fungerade.

En sak att fundera på är balansen mellan att dela upp arbetet så att man dels kan programmera självständigt och inte behöver kunna allt, dels att man får tillräcklig koll på resten av programmet så att man kan hjälpa till med andra delar vid behov och kan tillräckligt om hela programmet för att vara till nytta vid ihopsättningen. Om ett ansvarsområde visar sig bli för arbetsamt behöver det finnas någon som är tillräckligt insatt för att kunna rycka in. Det är dock svårt att veta när man delar ut ansvarsområden i början hur mycket arbete som kommer krävas för delar som man inte då vet så mycket om, i vårt fall till exempel det grafiska användargränssnittet.

Det var kul att göra ett projekt där man fick utnyttja sådant som redan finns skrivet, i mitt fall grafritarprogrammet, och inte bara som i labbarna göra allting själv. Det var också nyttigt att upptäcka att det finns hjälpmedel som SVN, Doxygen och WebCollab och att använda dem.