2012

Android Applikationen MyBar

REFLEKTION

ADAM CLARK, DAG FRIDÉN, MATHIAS BJURBÄCK, MATHIAS KARLGREN, VIKTOR EDSAND

CHALMERS TEKNISKA HÖGSKOLA | Software Engineering Project - DAT255

Innehållsförteckning

Introduktion 2
Androidutveckling 2
Eclipse 3
GitHub 3
Git 3
  Egit 3
  Git Shell 3
Scrum 4
  Problem och utmaningar 4
Kodstruktur 4
  Designmönster 4
    View 4
    Controller 4
    Model 4
  Paket se.turbotorsk.mybar 5
    se.turbotorsk.mybar.controller 6
    se.turbotorsk.mybar.model.* 6
Databas 7
  Content provider 7
  JSON - JavaScript Object Notation 7
  Extern kod 8
Gruppdynamik 8
  Möten 8
  Sharepoint 8
  Mailflöde 9
  Lync 9
  Mumble 9
Testning 9
  Robotium 9
  Content Provider 9
Framtid för projeket 9
Refaktorering 9
Handledning 10
Sammanfattning 10

Reflektion

Introduktion

Vi tycker alla att det har varit roligt med en praktisk kurs som bryter av den teoretiska jargongen på Chalmers. Däremot var det fortfarande mycket att lära sig med tanke på att ingen i gruppen programmerat i Android tidigare.

Att välja projekt var relativt enkelt, då alla i gruppen ville utveckla något som i våra ögon var användbart och roligt. En applikation som förhoppningsvis kan användas även utanför skolan och som attraherar en bred målgrupp.

Vi hade svårt att i förväg bestämma vilka funktioner som skulle vara med i applikationen och därför var det inledningsvis svårt att använda sig av Scrum på ett korrekt sätt. Detta i sin tur ledde till att vi hade svårt att prioritera det User Stories vi kommit fram till i inledningen av projektet.

En nyhet för oss var att använda ett system för versionshantering. Det gav oss både nya utmaningar men har också varit ett centralt verktyg i utvecklingsprocessen. Till en början förstod vi inte styrkan i att använda oss av versionshanteringssystemet Git och det blev snarare ett hinder än ett hjälpmedel. Vår inställning till Git förändrades allt eftersom att vi jobbade mer med det. Vi fick en förståelse om hur Git skulle användas genom att sitta och pröva sig fram.

Androidutveckling

Under projektets förlopp har vi gått från komplett okunskap inom Android, Eclipse, Git och JUnit till att en täckande kunskapsnivå där vi har kunnat utveckla en fungerande Androidapplikation.

Resan dit var kantad av diverse utmaningar som bromsade utvecklingen av funktionen i applikationen, speciellt under de första veckorna. Vi i gruppen tycker att det var svårt att komma in i Android programmering och det blev absolut inte lättare när man tidigare utvecklat med hjälp av varken Eclipse eller Git. Det gjorde att det tog lång tid för oss att komma in i utvecklingen av vår applikation och vi hade istället flera testprojekt för att komma in i API:t. Detta är anledningen till att det finns många branches som aldrig utvecklades till någon egentlig funktion i vår master branch.

Ett annat problem vi har haft är att få internet access till applikationen. Detta löste sig till slut men var något som tog tid. Saker som dessa syns inte som commits eller liknande och vi har alla jobbat hårt med problem som dessa.

Eclipse

På Lindholmen blev vi ombedda att använda BlueJ i stället för Eclipse. Detta för att lära oss grunden till OOP-språk som Java då våra första kurser var i ren C och assembler. Ett byte till Eclipse som IDE gjorde vi därför att vi ville använda det IDE alla andra använde sig av. Eclipse är också den utvecklingsmiljö Google själva rekommenderar.

Inlärningskurvan var till en början flack men efter många timmars arbete lärde vi oss mer och mer. Kortkommandon som var nyttiga att bemästra fick vi veta i 7:e veckan av projektets arbete. Formateringen av text underlättade mycket i slutskedet av projektet då vi lade ner mer tid på Java-standard och hanteringen av violations. Javastandarder och Androidstandarder med längd på kodrader var innan helt okänt, bland mycket annat.

GitHub

Vi valde GitHub som versionshanteringsprogram för att vi tidigare inte använt oss av något versionshanteringsprogram. Eftersom erfarenheter inom området inte fanns bestämmde vi oss för att följa handledarnas rekommendation och använda sig av GitHub. I efterhand är vi nöjda med valet, det var svårt att komma igång men om man jämför med andra alternativ så var detta det bästa.

Git

Vi tyckte att det verkade svårt att lära sig alla kommandon för att använda sig av terminalfönstret. Vi hörde oss därför runt om det fanns någon plugin till Eclipse som vi kunde använda oss av direkt i IDE:t. Egit dök upp tidigt och även där, som i fallet med GitHub, körde vi på första bästa. Egit var lätt att använda när man väl lärt sig grunderna, vilket dock krävdes någon vecka att komma in i. Även i detta fall är vi nöjda med valet.

Ett fel vi gjorde i Git mot slutet är att vi har commitade våra dokumentfiler. Fokuset låg i att knyta ihop projektet mer än att hantera text på optimalt sätt. Anledningen till att vi gjort på detta sätt är för att vara säkra på att det inte blir några krockar om två eller fler personer editerat samma fil. Optimalt är dock att sitta tillsammans med t.ex en projektor och spåna fram texten. Då undviker man krockar samtidigt som man får en högre kvalitet på texten.

Ett annat fel vi känner att vi gjort är att vi inte haft en ”development master”. I detta projekt merga:de vi in våra branches rakt in i mastern, vilket ibland skapade onödiga problem och krockar. Detta hade förmodligen undervikts om vi haft ett mellansteg mellan branches och vår master.

Egit

Majoriteten av gruppen har använt sig av tillägget Egit till Eclipse för att hantera vårat Git-repo. Detta har fungerat bra men var inte helt användarvänligt, och tog tid att lära sig. Dock är Egit ett väldigt smidigt verktyg för den som inte vill lära sig Git Shell.

Git Shell

Två av oss använde Git Shell istället för Egit. Verktyget har fungerat väl, men när det uppstod krockar fick vi också leta upp mergeverktyg m.m. Vid ett sånt tillfälle är Git Shell inte optimalt för nybörjare som oss.

Däremot var det inga problem att t.ex. byta branch, push, pull, commit, och diverse simpla uppgifter som vi primärt använt oss av.

Scrum

Ingen i vår grupp hade tidigare hört talas om varken Scrum eller Vattenfallsmodellen. Vi kände att Vattenfallsmodellen riktade sig mer till större projekt och valde därför SCRUM. Att veta allt man skulle göra inom Scrum och hur det hanterades, så som Sprint Planning, Sprint Backlog och Product Backlog, tog ett bra tag att förstå. Vi var tvugna att uppdatera utformningen på våra dokument nästan varje vecka. Det var alltid något som såg fel ut på grund av missförstånd eller bristande kommunikation mellan handledare och projektgrupp. Vi hade våra möten i början av veckan och tog upp vad som skulle göras. I slutet av veckan hade vi ett möte och summerade veckan som gått.

Undet projektets början insåg vi inte de många fördelar som fanns med att ha väldefinierade User Stories som innehöll information om hur programmet skulle fungera och vilka funktioner som skulle finnas med. Detta är något som vi i efterhand är överens om att vi skulle spara tid och resurer på att redan tidigt skapa dessa dokument, men även att hålla dem uppdaterade under projektets gång.

Problem och utmaningar

Då detta är första gången vi använt Scrum så har det varit svårt att se till att det finns User Stories för alla funktioner. Det finns några funktioner i vår applikation som tyvärr inte har en User Story. Det är något vi skulle sett till att det fanns om projektet gjorts om.

Vi känner dock att Scrum lämpar sig till bättre i något större projekt. Flera av dokumenten har vi skrivit endast för att följa Scrum-standard. Mer än att vi faktiskt behövt eller haft nytta av dem. Det känns värdefullt att ha följt Scrum-modellen, men det hade inte behövts i detta projekt.

Kodstruktur

Designmönster

Under projektet så har vi strävat efter att följa MVC-modellen. Programmet i sig har även en uppdelning i klient – server där den externa databasen kan ses som en server del och den lokala applikationen kan ses som klient.

Paketen delas upp enligt följande MVC struktur.

View

se.turbotorsk.mybar
(Paketet se.turbotorsk.mybar borde döpts till se.turbotorsk.mybar.view.)

Controller

se.turbotorsk.mybar.controler

Model

se.turbotorsk.mybar.model
se.turbotorsk.mybar.model.database
se.turbotorsk.mybar.model.externaldata

Om man kollar i våra UML-diagram kan man se att två klasser i View har kännedom om två klasser i model. Detta är något som inte var tänkt från början men på grund av tidsbrist i slutet av projektet hade vi inte tid att göra nödvändig refaktorering och ta bort de kopplingarna.

Paket

se.turbotorsk.mybar

Detta paket innehåller kod som har med design och view att göra. Till exempel finns här vår MainActivity som är grunden till programmet.

Till en början ville vi använda oss av fragments, detta för att få möjligheten att snabbt kuna förflytta oss i applikationen. Vi lade ner väldigt mycket tid på att försöka få detta att fungera men misslyckades med det. Efter mycket om och med bestämde vi oss för att istället använda oss av en tab layout som är föregångaren till swipe. Det blev då mycket enklare att komma vidare i applikationsutvecklandet och kunde lägga tid på annat.

Då man använder sig av tabs så behöver man inte använda fragments, det var just denna del vi hade svårt med. Vi hade alltså väldigt gärna velat använda fragments men tyvärr fick vi inte detta att fungera. Självklart är detta ett perfekt fall för framtida refaktorering.

Under de första veckorna blev vi tvungna att läsa på extremt mycket för att får en uppfattning på hur layouter fungerar och hur man kopplar dessa till en Android applikation. Till en början utecklade vi många GUI-funktioner själva som vi sedan insåg att de redan fanns i det API som Android tillhandahåller.

Därför blev det många branches som skapades utan att de har blivit merge:ade in i vår master. Vi hoppas att ni har dett i åtanke.

Designmässigt är vi nöjda med vår applikation och tycker att den ser bra ut. Där emot finns det självklart mer att göra, men så är det alltid. Valet till att använda oss av många listor är självklart med tanke på vad vi vill uppnå med vår applikation. Vi märkte snabbt att en vanlig lista blev för tråkig och simplel, vi valde däröfr att göra vår egna. Vi utvecklade ett eget listelement (rowlayout) för att få den att se ut som vi ville, med bilder, rating, ingredienser med mera.

Just eftersom att vi använde oss av en snyggare listvy för våra drinkar, än den som är standard, så blev vi tvungna att utveckla en egen adapter för att lättare hantera den data en drink innehöll. I ett senare skede fick vi också göra en adapter för ingredienser som är snarlik den för drinkar.

Att skapa en layout och adaptrar till dessa var extremt tidsödande och krävde stor förståelse för hur Androids GUI hanteras. Vi känner inte riktigt att den research vi gjort för att få en snygg layout speglas i koden då man nu kan tycka att det var enkel kod. Dock tyckte vi verkligen inte detta de första veckorna!

En annan detalj i applikationen vi hade problem med var options-menyn (den som visas i form utav tre stycken kvadrater staplade på varandra). Vi läste på mycket om hur det fungerar och hur menyerna inuti skulle skapas, och fick fram ett resultat efter några dagar. Senare insåg vi att den visades på olika ställen/sätt beroende på vilken telefon man använde, eller om man körde i emulator på datorn. Detta problem tog även det lång tid att lösa, men till sist kunde vi göra så att den i alla fall visade sig på allas telefoner och även i emulatorn. Var någonstans den däremot visas är fortfarande ett mysterium som vi inte kunnat åtgärda. Därför ser det fult ut på vissa telefoner sominte använder sig utav en ”fysisk” options-menu-knapp på telefonen, utan befinner sig i en grå rad längst ner på telefonens skärm.

Vi, Adam, Mathias B och Viktor, blev tvungna att lägga ner mycket tid på att försöka förstå hur layouter byggs upp innan vi ens kunde göra något konkret. Det vi trodde skulle vara enkelt blev istället en av våra största utmaningar.

Att förstå java kod är en sak, det har vi gjort tidigare, förstå xml är en helt annan. Att sedan väva samman dessa två standarder var inte det lättaste.

Det krävdes många timmars tålamod innan vi förstod hur man använde sig av R filen ordentligt och på så vis kunde binda xml filer till kod.

se.turbotorsk.mybar.controller

Genom att använda ett Controller objekt kan vi undvika att View känner till hur data hanteras i Modell. Många av metoderna i Contoller skickar enbart vidare data från Data.java till de motoder i View som behöver dem. Vi insåg sent i projektet att vi hade två klasser i View (DrinkAdapeter.java och IngredientAdaper.java) som känner till klassen Drink.java och Ingredient.Java i Model. Det skulle innebära för mycket refaktorering i Model för att vi skulle kunna eliminera dessa beroenden på ett bra sätt i slutet av projektet. Deta är något som kommer att genomföras om vi väljer att forstätta utvecklingen av applikatioen.

För varje program kommer enbart en Controller att finnas och därför valde vi att sätta den till typen final static. Detta gjort det även det lättare att göra anrop till metoderna från övriga klasser och packet.

se.turbotorsk.mybar.model.*

Vi hade redan i projektes inledning en ganska tydlig idé på hur vi skulle hantera vår data. Denna idé gick ut på att ladda in informationen från en extern databas. Informationen skulle hämtas direkt med SQL när programmet installerades, eller startades för första gången. Datan skulle därefter sparas till en lokal textfil.

När vi undersökt om det fanns någon möjlighet att använda SQL i Android, insåg vi att detta inte var någon optimal lösning. Designen medförde säkerhetsrisker och kunde göra applikationen känslig för uppdateringar av till exempel SQL-programvaran på den externa servern. Problem kunde även uppstå när det gällde uppdateringar av databasen samt när nya drinkar och ingridienser skulle läggas till.

Genom att använda ett skickt mellan applikation och databas kunde vi undervika de allvarligaste säkerhetsriskerna. Detta skickt består av en extern kod skrivern i PHP som låter skapa en HTML-fil. HTML-filen innehåller en så kallad JSON struktur, vilket representerar Drink- och Ingrediensdatan. Detta gör att applikationen inte längre har någon möjlighet att utföra förfrågningar direkt mot vår externa databas. Vi kan även göra sidan är statisk under vissa perioder med hög belastning, och på så vis minska belastningen på den externa databasen.

Den största utmaningen med att populera databasen med information var att konvertera ett recept med innehåll sådant att applikationen kunde tolka detta på ett bra sätt. Målet var att fylla databasen med 100 drinkar, detta uppnådde vi dock aldrig. Det blev för mycket jobb att konvertera drinkarna, vi bestämde oss istället att lägga krut på applikationen. Istället för att slösa tid på att göra något vi förstått.

Processen var omfattande på så vis att vi behövde ha informationen i ett CSV format som i sin tur representerar drinken så vi vill. Detta är något som är möjligt att automatisera med mer extern kod men vi kände inte att det var aktuellt med projektets tidsram.

Databas

Content provider

MyBar var designad runt tanken att vi alltid är uppkopplade mot molnet. Därför är den lokala databasen tom vid start och en automatisk hämtning av data sker från vår externa databas över JSON network calls. Vi tänkte med automatik hämta systembolagets XML-fil med alla deras produkter och parsa det till vår remote data store. Den externa servern skickar en förfrågan till mobilklienterna om de vill uppdatera och en sync sker till den lokala databasen.

Efter granskning av sätten att prata med databaser i android API:t valde vi Content Provider framför att prata direkt med vår lokala data store. Med en Content Provider hade då andra mobilapplikationer kunnat använda våra tabeller och vi hade kunnat göra fler appar som använder MyBars Content Provider. Vi alla gillar att dela upp allting i mindre objekt som inte känner till varandra och att strikt följa MVC-modellen. Content Provider stämde överens med vårat sätt att tänka, samt fungera som ett extra lager mellan databasen och applikationen i övrigt. Vi ville stark undvika att prata direkt med raw SQL queries. Providern exponerar tabellerna på ett standardiserat sätt mellan applikationer och kontrollerar att krockar ej sker om flera appar sänder frågor samtidigt. Vi får även tillgång till funktioner vi annars inte haft som kopiering av komplex databasdata mellan applikationer. Vi slipper också att tänka på malicious SQL queries från andra applikationer då Providern tar hand om detta.

När vår applikation vill prata med vår data store får vi använda en Content Resolver för att hämta data med CRUD (insert, update, delete, query), men de är endast abstrakta och implimentationen av dessa hämtas ur Content Providern. Det var väldigt svårt att komma igång då jag var tvungen att lära mig nya främmande konkept och få en förståelse för hur Content Provider, Content Resolver och SQLite Database Helper objekten kommunicer med varandra.

Jag ångrar mig inte att vi skapade en Content Provider trots att det var mycket svårare än att gå den lätta vägen med att skapa en SQLite Database Helper och sända SQL queries direkt in i den. Därför att jag lärde mig väldigt mycket om Content Providers och hur testning av denna fungerar.

JSON - JavaScript Object Notation

Det fans många olika guider och example på hur man kunde använda JSON-klasser för att tolka JSON och översätta datan till en ny lagrinsform. Det var dock svårt att hitta ett bra example som gav oss ideér på hur vi kunde implementera JSON i just vårt projekt. För att motverka problem och göra det lätt för oss att skriva koden valde vi det JSON klass som finns inbyggt i Andriod. När väljer att använda de inbyggda klasserna gör vi applikationen mindre plattformsoberoende. Detta gör att det kommer krävs en refktorering av JSON klassen applikationen skall portas till annan platform.

Extern kod

På vår externa server finns en MySQL databas och en Apache webserver med PHP installerat. I PHP finns både funktioner för att hämta data samt konvertera den till formatet JSON. Därför var det naturligt för oss att använda PHP.

I den första versionen av klassen skpades ett PHP script som läster datan från MySQL och sedan skapade ett JSON array med hjälp av en loop som vi själva skpat strukturen för. Det uppstod stora problem med denna metodt då många tecken inte formaterades korek. För att lösa detta gick vi över till att använda redan i PHP redan befintliga metoder. Detta löste problemet med skräpdata och gjorde koden avsevärt mycket snyggare.

Datan hämtas med en standard HTTP-get. Även detta får vi hjälp med med från inbyggda klasser i Android. Under projekts slutfas kom frågan upp angående hur vi kan göra så att inga obehöriga får tillgång till vår information. Vi kom fram till att detta går att lösa genom att gå från HTTP till HTTPS samt skicka med två varibaler i den web-förfrågan som enheten skickar för att få tillgång till datan. Dessa variabler matchas mot två variabler i webservern, stämmer variablerna överrens returneras en giltlig JSON kod, är det fel variabler retuneras en tom sida.

Gruppdynamik

Under projektets gång har vi använt oss av flera olika sätt att kommunicera, hantera, administrera och utveckla på. Detta är något vi är väldigt nöjda med. Det har under projektetes gång inte varit några som helst problem att få kontakt med resten av gruppen. Vi har enkelt kunnat förmedla information och kod med hjälp av de verktyg vi har valt att använda. Dessvärre har inte den externa kommunikationen fungerat tillräckligt väl.

Möten

Vi har varje söndag haft ett långt möte där vi varje vecka har gått igenom vad vi gjort, och vad vi ska göra för att komma framåt. Vi har också gått igenom de User Stories och ärenden vi haft. På dessa möten har vi också diskuterat till exempel layouter och hur kopplingar till databaser skall se ut. Vi har gemensamt kommit fram till beslut som sedan genomsyrat hela projektet.

Innan vi började programmera bestämde vi på ett av våra första möten att lägga ner mycket tid på att ta reda på information. Detta för att behöva göra om så lite som möjligt.

Under veckorna har vi träffats i grupprum för att utveckla och diskutera applikationen tillsammans, det är lättare att komma fram till beslut om man ses i person. Det är på dessa möten vi kommit fram till många avgörande beslut i utformningen av applikationen och projektet i sig.

Vi har också haft möten med vår handledare, mer om dessa möten under rubriken handledning.

Sharepoint

Under de första veckorna försökte vi oss på att använda Microsofts produkt Microsoft Sharepoint. Programmet användes för att hantera dokument och projekt i allmänhet. Tanken var att vi med detta program skulle ha allt på en plats. Allt eftersom projektet fortskred så fasades Sharepoint ut. Vi kan än idag inte säga varför det blev så, men mailflödet tog över kommunikationen och Git tog över dokumenthanteringen allt eftersom att vi lärde oss Git ordentligt. Vi känner dock att detta är en produkt som är optimal vid större projekt som på företag eller längre projekt på Chalmers. Vi hade gärna setta att vi använt oss av till exempel Google Drive för dokumenthantering.

Mailflöde

Från dag ett skapade vi ett mailflöde som vi egentligen skulle använda som komplement till Sharepoint, detta tog dock över som kommunikationskanal då det blev extremt smidigt att kunna maila fram och tillbaka i till exempel telefonen. Flödet har använts frekvent under hela projekttiden och vi är nu uppe i drygt 500 mail. Endast det i sig är ett tydligt exempel på att vi har haft bra kommunikation inom gruppen.

Lync

Vi har under projektet använt oss av proramvaran Lync för att dela skärmar med varandra och arbeta tillsammans.

Mumble

Mumble är den mjukvara som användes då vi hade telefonmöten över internet. Det har fungerat bra och har varit ett givande komplement till face-to-face möten.

Testning

Robotium

Se separat bilaga.

Content Provider

För att testa Providern extendade jag ProviderTestCase2 som förser oss med en MockContentProvider som imiterar en Content Provider. Då sker alla testar mot Providern i en isolerat Context. Hade vi i stället testat med den vanliga Content Providern hade oönskade ändringar skett i databasen, speciellt om inte testcasen gått igenom korrekt. Eftersom vi använder en MockedContentProvider behövde jag inte lägga till något i tearDown() utöver det ProviderTestCase2 gör. MockContentResolver klassen isolerar testerna helt från det riktiga innehållet. Inga ändringar i manifestet har heller varit nödvändiga.

Framtid för projeket

Alla i projektgruppen är intresserade av att fortsatta utvecklingen av appen. Mestadels av jobbet som återstår är att uppdatera applikationen med mer drinkar och ingredienser. Gruppen måste även bygga bort ett par av de begränsningar och mindre buggar som finns kvar i version 0.7. När detta är utfört måste stöd för Google Licensning och bättre felhantering byggas in.

När de mer tekniska bitarna är utförda och på plats måste ett nytt Google Play konto skapas och gruppen komma överrens om hur det framtida samarbetet ska se ut.

Refaktorering

Under projektets gång har vi flera gånger refaktorerat. Detta för att få en bättre struktur på vår kod. I början av projektet hade vi till exempel ingen controller och vi anropade model direkt. En stor refaktorering var då att implementera den Controller klass vi ville anävnda.

Vi refaktorerade också det grafiska gränssnittet för att få appen som vi ville ha den. Från början skulle vi använda oss av fragments och swiping. Detta bytte vi till tabs och fick senare reda på att detta följer Android standard när målet är att visa en sida i taget.

Handledning

Vi har känt att Android utveckling var trögstartat men, vi känner nu att vi förstår en hel del. Något som vi verkligen skulle se från Er sida är att lägga mer krut på att möjligtvis ha en föreläsning om hur man utvecklar ett snyggt UI och till exempel sätter samman xml och Java kod. Detta för att på ett snabbare sätt få in den information som krävs för att komma igång att koda.

Visst, det finns guider på nätet och det var dit vi gång på gång blev hänvisade. Men ni som lärare måste ta ett större ansvar och ge oss relevant information i form av till exempel obligatoriska föreläsningar. Vi kände, tyvärr, ingen röd tråd i kursen och planeringen stämde aldrig. Det blev väldigt lätt att föreläsningarna svävade ut i andra områden än aktuella. Kursen behöver skäras ned och kokas ihop till en mer väldefinierad kurs.

Det var många gånger man inte fick ett svar som hade någonting med frågan att göra. Det kändes som att vi inte fick ett svar på våra frågor för att handledaren helt enkelt inte hade ett bra svar. Det var många gånger vi blev hänvisade till internet och det är bra, till en viss grad.

Vi fick det väldigt svårt att komma vidare med projektet utan en handledare som fördstod problemet. Vi tror dock att handledningen skulle bli bättre om kursen var bättre planerad.

Det kändes väldigt oproffsigt även om det kanske är så att ni ser det som att vi lär oss mycket på att leta fakta. Vi skulle i sådana fall behövt någon vi skulle kunnat verifiera våra fynd. Detta måste finnas.

Vi skulle gärna se en mer genomtänkt handledning där vi faktiskt kan få hjälp med de krav ni har satt upp. Det var flertalet gånger vi fick tvetydiga svar vecka efter vecka och vet fortfarande inte om vissa dokument ser ut som de skall. På exempelprojektet som finns på Git finns inte heller alla de dokument som vi detta år ska skicka in, därför är inte detta ett hållbart argument till varför handledaren skickade oss dit för information.

Vi fick gång på gång svar att handledaren inte haft tid att läsa dokumenten. Detta ser vi som förvånande då handledare får betalt. Flertalet gånger kom vår handledare på möten utan att ha uppdaterat repot, ifrågasatte varför inte dokument var inlämnade, uppdaterade repot och såg att dokumenten var där.

Vi vill inte att ni skall ta detta personligt, men vi känner att det är viktigt att framföra vilka problem kursen har i sig, och inte projektet. Det behövs en rejäl funderare över till exempel betygskrav, det känns inte speciellt proffsigt att ändra dessa krav under kursens gång.

Sammanfattning

Projektet har varit mycket utmanande och man har tvingats att använda många nya verktyg, arbetsmetoder och en helt ny plattform för utveckling. Projektet har även involverat utmaningar inom gruppsamarbete. Dessa är utmaningar som alltid kommer att inträffa under ett projekt som involverar flera personer eller intressenter. Varje tillfälle som ges att utveckla kunskapen inom just det området är speciellt viktigt för oss unga ingenjörer som snart skall ut i arbetslivet.

Nu i slutet av projektet är alla i gruppen nöjda med utfallet av vårt arbete, men alla är samtidigt överens om att det finns en stor förbättringspotential i kursens upplägg. Något som vi hoppas sker till nästa år för att ge nästaårs elever en bättre helhetsupplevelse.