Post-Mortem Report

Chalmers tekniska högskola
Software Engineering Project
Handledare: Morgan Ericsson
2014-06-05

Ludvig Börjesson
Pauline Daremark
Gabriel Kamienny
Peter Jonsson

Innehållsförteckning

1 Reflektion utvecklingsmetoder och tekniker

Vid utveckling av programvara finns det två huvudsakliga metoder. Den traditionella tillämpningen är vanligtvis vattenfallsmetoden, vilken utgår från att allt görs sekventiellt efter en redan färdig produktspecifikation. Denna metod kräver mycket arbete redan innan det går att börja utveckla mjukvaran. De fem stegen med kravspecifikation, design, implementation, kvalitetssäkring och vidare utveckling ses som ett flöde. Dessa delar utförs i sekventiell ordning och upprepas därefter inte. Detta sätt att konstruera och utveckla saker kommer från början från byggindustrin, och anpassades till mjukvaruutveckling eftersom det inte fanns några andra kända principer. Flertalet gånger har det dock visat sig att detta inte är någon bra metod för programvaruutveckling, fler och fler har därför istället börjat använda sig av agila metoder. Det som kännetecknar agila metoder är främst ett nära samarbete med kunden och ett iterativt och inkrementellt arbete. Istället för att leverera en färdig produkt till kund prioriteras korta arbetscykler och i slutet av varje cykel får kunden tycka till om leveransen. Med agila metoder behöver utvecklaren inte veta hur slutprodukten ska se ut från början, utan arbetar efter kundernas önskemål och krav samt mjukvarans vision.
När vi skulle börja detta projekt var det givet att vi skulle arbeta agilt och inte inkrementellt enligt vattenfallsmetoden. Vi upplevde att ett agilt arbetssätt var en nödvändighet, framför allt eftersom det från början inte var helt givet hur den slutgiltiga produkten skulle se ut. I och med att våra tidigare erfarenheter inom mjukvaruutveckling är begränsade var det heller inte helt givet hur vi skulle hantera uppgifter, utveckling av appen, varför ett agilt arbetssätt än mer motiverades. Nedan presenteras de agila metoder och tekniker som vi försökt tillämpa samt reflekterat över under utvecklingsperioden.

1.1 Scrum

Scrum är en utvecklingsmetod som kännetecknas främst av att arbetet delas upp i user stories som i sin tur delas in i sprintar, vilka kan vara mellan tre och 30 dagar. En user story är en mening i vardagsspråk som förklarar vad användaren behöver för att genomföra det som krävs i sammanhanget. Ett utvecklingsteam delar sedan kontinuerligt upp alla user stories mellan sig i olika sprintar. Varje sprint ska inledas med en planeringssession och avslutas med sprintgranskning och en förbättringsaktivitet som kallas retrospective eller återblick på svenska. Varje dag ska ett kortare daily scrum eller stand-up meeting hållas då samtliga projektmedlemmar uppdateras om projektets läge. Jämfört med traditionell utveckling är gruppmedlemmarnas olika roller betydligt färre då det vid scrumutveckling endast finns produktägare, scrum master och utvecklingsteam.

1.1.1 Våra erfarenheter

Vi tror att scrum är en bra metod att tillämpa vid utveckling av mjukvaruprojekt. Men av flera anledningar passade metoden inte oss helt perfekt, varför denna metod inte tillämpats fullt ut. En aspekt som försvårade ett arbetssätt helt i linje med scrum var avsaknaden av en fysisk kund att leverera mjukvaran till. Med anledning av detta fick vi hela tiden sätta kraven själva, och hade inga tydliga deadlines för leverans, vilket gjorde att sprintarna hade en tendens att flyta ihop.

En ytterligare anledning till att scrum inte tillämpades fullt ut var tidsaspekten. Eftersom alla fyra gruppmedlemmar arbetade med kandidatarbetet parallellt med detta projekt så kom vi inte igång med projektet på allvar förrän efter kandidatarbetet var inlämnat i början av maj. Detta har medfört att istället för att sitta åtskilda, och varje vecka arbeta med en normal arbetstakt på halvfart, har vi de sista veckorna för det mesta arbetat tio till tolv timmar om dagen. Vi har också kunnat sitta mycket tillsammans, vilket har inneburit att det inte funnits samma behov av daily scrum-möten. Detta eftersom vi då ändå haft bra insikt i vad övriga gruppmedlemmar arbetat med och nästan alltid haft möjlighet till att ställa frågor rakt ut. Med tanke på att tiden till inlämning var begränsad och att vi inte hade någon tydlig uppfattning av hur lång tid utvecklingen skulle ta så valde vi att inte sätta oss in i alla aspekter av scrum. Vi prioriterade istället att komma igång med kodandet och att fräscha upp våra kunskaper inom javaprogrammering, eftersom vi var oroliga att vi annars inte skulle få ihop en slutprodukt. Med tanke på den tillgängliga tiden och de korta, intensiva sprintarna fokuserade vi inte heller speciellt mycket på retrospective eller sprintplanering. När det gäller roller utsågs Pauline till både scrum master och projektägare, då det i vår lilla grupp kändes överflödigt att ha olika personer till detta. Detta innebar att hon hade huvudansvaret för utvecklingsprocessen, därmed ett övergripande ansvar för allt som gjordes. Hon hade även ansvar för att försöka se arbetsflödet och applikationens funktioner ur kundens synvinkel.

1.1.2 Styrkor

Den största styrkan med scrum som vi ändå upplevt var just sättet att strukturera och dela upp arbetet. Vi började med att brainstorma och dela upp produkten i olika övergripande områden eller epics, som är stora user stories som inte kan fullföljas under en sprint. Sedan gavs en eller två personer övergripande ansvar för olika övergripande områden eller epics, vilket var bra för att veta ungefär vad varje person skulle göra, även detta var långt i från en benhård uppdelning. Därefter började vi fundera och skriva user stories för grundläggande funktionalitet. Det var sedan väldigt praktiskt att utgå från user stories när arbetet delades upp mellan utvecklarna. Genom att varje person valde user stories, visste man vilken funktionalitet man skulle jobba med och implementera. Den dagliga kommunikationen som scrum poängterar vikten av har även vi upplevt fördelarna med. Även om vi inte i någon större utsträckning använt oss av daily scrum-möten har vi ändå haft möjlighet till en kontinuerlig kommunikation, eftersom vi allt som oftast suttit tillsammans. När någon haft problem har denne konsultat någon annan gruppmedlem och på så sätt fått hjälp med att lösa problemet.

1.1.3 Svagheter

Den största svagheten med scrum ur vår synvinkel var känslan av att denna metodik framför allt lämpar sig för större och längre projekt med specifika kunder. Även om tröskeln att börja med scrummetodik är betydligt lägre än för inkrementella metoder upplevde vi, vilket tidigare redogjorts för, att det var allt för tidskrävande att sätta sig in i detta arbetssätt och sedan tillämpa detta fullt ut. Med tidsramen och omfattningen av vårt projekt kändes inte heller alla steg relevanta. Eftersom huvuddelen av vårt projekt var väldigt intensivt och sprintarna blev väldigt korta (i genomsnitt fyra dagar), var det inte alls relevant att ha omfattande planering, återblickar och sprintgranskning i början respektive slutet av varje sprint.

Vi testade däremot att ha några officiella daily scrum-möten. Men dessa kändes mest överflödiga och konstlade, eftersom alla ändå var väl medvetna om vad som pågick och hur utvecklingen fortskred.

1.1.4 Reflektioner inför framtida projekt

Även om vi långt ifrån tillämpade scrum som den metodik som Jeff Sutherland och Ken Schwaber utvecklade har vi fått många positiva vibbar från scrummetodiken. I mer normala projekt, som inte är lika intensiva och där utvecklarna inte har kontinuerlig kommunikation, tror vi att scrum är en väl fungerande metodik. Just kommunikation är något som vi verkligen har insett är viktigt vid mjukvaruutveckling. Om inte samtliga i gruppen har översiktlig koll på hur utvecklingen fortskrider kan detta resultera i mycket onödigt arbete och till och med att kontraproduktivt arbete utförs. Därför tror vi bland annat att daily scrum-möten är ett utmärkt sätt för att upprätthålla en kontinuerlig kommunikation, så länge inte utvecklarna hela tiden sitter i samma rum och kan kommunicera fritt ändå. En ytterligare reflektion som vi gjort är att avsaknaden av faktiska daily scrum-möten antagligen skulle få mer förödande konsekvenser om antalet gruppmedlemmar hade varit större. I detta fall, då gruppen endast bestått av fyra personer, har den kommunikation som förts kontinuerligt varit tillräckligt. Däremot skulle antagligen en mer uppstyrd kommunikation varit nödvändig om antalet varit större, eftersom antalet kommunikationsvägar och kravet på informationsutbyte då ökar stort.

I större projekt skulle vi även välja att inleda varje sprint med en sprintplanering och avsluta med sprintgranskning och retrospective. Något som vi också skulle tillämpa i större projekt skulle vara att definiera alla user stories i story points. I vårt projekt kändes det inte värt tiden att göra detta och spelade inte så stor roll, utan så fort någon gjorde klart en user story så började personen direkt på nästa. Ska arbetet däremot fördelas jämt under en längre tid tror vi att användningen av story points är ett bra förfarande för att uppnå detta. Genom detta går det antagligen att lättare få en överblick av hur projektet ligger till, och arbetsbördan blir jämnare bland utvecklarna. Denna överblick om hur vi låg till var något som vi saknade under projektets gång. Om vi hade kommit i gång tidigare hade det förmodligen varit värt att prioritera en tilldelning av story points. Bland annat hade detta förmodligen inneburit att oron över huruvida vi skulle bli färdiga i tid eller inte hade kunnat minskas.

Något annat som antagligen också hade inneburit att gruppmedlemmarna fått en bättre överblick hade varit om en person givits ett större övergripande ansvar för projektledningen, och mindre ansvar för de delar som inkluderar faktisk mjukvaruutveckling. Vi hade helt klart kunnat strukturera upp och försöka få en bättre förståelse för vad rollerna som scrummaster och produktägare innebär, för att på så sätt ha kunnat dela upp arbetet på ett mer optimalt sätt. Ingen av gruppmedlemmarna hade någon större erfarenhet av mjukvaruutveckling innan projektet, vilket gjorde att vi trodde att samtliga medlemmar skulle behövas till själva kodningsprocessen. Detta gjorde att fokusen på scrum blev mindre och att vi fick en sämre överblick av utvecklingsprocessen. Om projektet gjordes om från början hade en person fått ett mer övergripande ansvar, och inte så mycket ansvar för själva mjukvaruutvecklingen. På så sätt hade vi nog även kunnat dra lite mer nytta av fördelarna med scrum.

1.1.5 Slutsats

Vår slutsats efter detta projekt är att kommunikationen är A och O vid mjukvaruutveckling, eftersom alla delar hänger ihop och bidrar till helheten. Eftersom vi hade möjlighet att sitta tillsammans majoriteten av tiden vi programmerade så gjorde detta att vi klarade oss bra, även fast vår tillämpning av scrummetodiken var bristfällig. Vid större mjukvaruprojekt tror vi dock scrum är en ypperlig metod för att undvika kommunikationsproblem.

1.2 Extreme programming (XP)

En annan agil metod som det redogjorts för på föreläsningarna är extreme programming, där det läggs stort fokus på kommunikation, enkelhet, återkoppling och respekt. Utifrån dessa aspekter finns tolv olika tillämpningar, av vilka vi valt att tillämpa några. Enligt namnet är det ett extremt sätt att programmera, och det är bara meningen att den praxis och de tekniker som passar det aktuella projektet som ska tillämpas. Orsakar praxis istället problem ska dessa delar inte tillämpas, vilket vi har utgått från.

1.2.1 Våra erfarenheter

Den teknik från XP som vi främst har tillämpat är parprogrammering. Denna teknik har tillämpats under större delen av projektet, framför allt då vi utvecklat de delar som vi upplevt som mer avancerade och som vi inte var bekanta med sedan tidigare. Anledningen till detta var att det då gavs möjlighet till att diskutera olika lösningar och att bolla olika idéer. För den person som faktiskt satt vid tangentbordet upplevdes tillämpningen av parprogrammering även som betryggande, detta eftersom det då fanns en ytterligare person som hela tiden kontrollerade koden som faktiskt skrevs. Ett exempel på detta var då vi började kolla på hur vi skulle koppla upp vår applikation mot Microsoft Azure samt vid implementeringen av en kontaktlista med hjälp av SQLite.

En ytterligare praxis som vi tillämpat är gemensamt ägandeskap. Vid denna typ av mjukvaruutveckling, där en förutsättning varit att utvecklingen sker snabbt och att applikationen tillåts att hela tiden ändras och uppdateras, är det viktigt att alla känner sig bekväma med att göra förändringar i redan existerande kod. I linje med detta har vi även ägnat oss åt kontinuerlig integration och versionshantering. Då ett visst kodavsnitt har ansetts som färdigt och väl fungerande har detta avsnitt synkats ihop med resterande kod, vilket har skett genom GitHub. Detta har fungerat bra och möjliggjort att flera personer har kunnat arbete med olika kodavsnitt samtidigt.

1.2.2 Styrkor

Extreme programming har många bra principer, som parprogrammering och gemensamt ägarskap. Vi hade inte möjlighet att testa alla med tanke på vårt projekts omfattning. Sedan är det en styrka att man faktiskt inte måste följa alla tolv tillämpningar om man inser att några av dem orsakar problem. XP har många likheter med Scrum, i synnerhet närheten till kund, vikten av kommunikation och små, kontinuerliga leveranser. Styrkor med XP jämfört med Scrum är att det finns fler riktlinjer för hur koden ska se ut och byggas upp. Scrum styr mer utvecklingsprocessen, vilket skulle kunna leda till att olika utvecklare i större utsträckning skriver på olika sätt. I XP däremot finns tydliga riktlinjer för hur koden ska byggas upp, exempelvis att den

ska vara väldigt enkelt och ska kontinuerligt struktureras om. Att det finns principer att följa för hur koden ser är bra för någon som inte ägnat sig så mycket åt mjukvaruutveckling.

1.2.3 Svagheter

En stor svaghet med XP är dock den höga tröskeln och det som krävs för att man ska kunna tillämpa metodiken fullt ut. Metodiken fungerar säkert för stora projekt, men för mindre projekt är det mer tveksamt. För oss var det ur flera aspekter helt omöjligt att tillämpa alla principer inom XP. Enligt metodiken för testning ska varje rad kod testas innan den skrivs, vilket var helt omöjligt för oss att åstadkomma med vår tidsaspekt och kunskap. Att all kod ska skrivas av par är inte heller något som gått att tillämpa fullt ut, även om de varit väldigt givande då det gjorts. En annan tillämpning inom XP är att kunden hela tiden ska vara på plats, vilket vi tror är bra, men detta var helt omöjligt att åstadkomma för oss då ingen uttalad kund fanns. Vi fallerade också på principen att man bara får jobba 40 timmars veckor, något som vi dock tror är en bra och sund värdering som sannolikt i längden ger goda resultat.

1.2.4 Reflektioner inför framtida projekt

Det är svårt att se att vi någon gång kommer kunna tillämpa XP fullt ut, då det skulle krävas att vi var delaktiga i relativt stora mjukvaruutvecklingsprojekt. Däremot har metoden många tillämpningar som verkligen är vettiga att ha i åtanke, även om inte alla kan appliceras. Parprogrammering är något som vi upplevt att det fungerat bra, som vi skulle fortsätta att applicera, men förmodligen inte hela tiden. Även enkel design är något som vi skulle tillämpa i större utsträckning. Under detta projekt började vi att göra det väldigt smidigt för oss och lägga majoriteten av koden i samma aktivitet. I slutet av projektet kom vi sedan på att det gjorde allt väldigt rörigt och svårt att förstå för någon utomstående, och vi började dela upp informationen på mindre klasser. Om vi haft ett större fokus på enkelhet redan från början tror vi att detta hade underlättat, och vi hade sluppit en hel del merarbete i slutskedet av projektet.

1.2.5 Slutsats

XP är en sund metodik för utveckling av mjukvara, men det är svårt att applicera metodiken i sin helhet. Däremot kommer vi se de olika tillämpningarna mer som tips eller verktyg som kan användas när en annan metodik som scrum appliceras. Grundpelarna är desamma i både XP och scrum, och vi tror att ett projekt hade utformats relativt likartat oavsett vilken av metoderna som använts. Förmodligen hade XP gett bättre kod, eftersom det finns mer regler för hur koden ska byggas upp och testas, till skillnad från scrummetodiken som främst beskriver arbetsprocessen. Tröskeln att börja med XP är dock så pass mycket högre att vi tror att det inte är värt att tillämpa metodiken om inte projektet är tillräckligt stort.

2 Reflektion jobba i team

Detta projekt i software engineering har genomförts i grupper om fyra personer, vilket kändes som lagom många med tanke på projektets upplägg och omfång. Det hade dock kunnat vara en person till, men då hade denna endast fått ägna sig åt projektledning och dokumentation, så att resterande fyra kunnat fokusera ännu mer på själva mjukvaruutvecklingen. Nu fick Pauline först i slutskedet av projektet huvudansvar för dokumentationen, medan de andra fortsatte koda med olika inriktningar.

2.1 Tidsåtgång

I början av kursen fungerade gruppen lite sämre, men detta var mycket på grund av att vi alla samtidigt skrev kandidatarbete i tre olika grupper. Det var svårt att hitta tid att komma igång, och en ganska hög tröskel innan vi kunde komma igång med programmerandet. Flera gruppmedlemmar var lite oroliga i detta skede av projektet. Men när kandidatarbetet väl var inlämnat kunde vi börja fokusera på mjukvaruutvecklingen på allvar. Då började också gruppen fungera mycket bättre. De tre sista veckorna har vi förutom dagtid också ägnat kvällar och helger åt att programmera tillsammans. Detta har gett resultat, och vi är positivt överraskade och stolta över vad vi lyckats åstadkomma. Går man in på timmar tror vi samtliga gruppmedlemmar har lagt ner nästan lika mycket tid. Läsvecka 1-2 har varje person i snitt lagt ner runt sex timmar, läsvecka 3-5 i snitt åtta timmar, läsvecka 6 cirka 30 timmar och de sista veckorna 7-8 har vi legat på ungefär 50 timmar i veckan uppskattningsvis har minst 30 timmar ägnas åt att redovisning och slutgiltig dokumentation de sista två veckorna. I denna tid inkluderas föreläsningar, handledningar, programmering, möten, dokumentation, testning etc.

I och med att vi ofta har använt oss av parprogrammering har vi under projektets gång stundtals upplevt att detta arbetssätt har inneburit att värdefull arbetstid gått till spillo, i och med att en och samma uppgift upptagit två olika personer. I efterhand tror vi dock att detta arbetssätt har inneburit att arbetet blivit mer tidseffektivt. Detta beror på att parprogrammering upplevts innebära att mer lämpliga programmeringsbeslut tas från början, vilket innebär att mindre tid behövt läggas på felsökning och omarbetning av koden.

Dessutom har parprogrammering (och även uppdelning av uppgifter på individnivå) upplevts som tidseffektivt då detta innebär att det endast är de personer som är ansvariga för en funktion som behöver ha full förståelse för hur implementeringen av denna ser ut. Detta har inneburit att vissa personer blivit mer specialiserade inom vissa funktioner, varför dessa personer fått hantera förändringar och oklarheter i dessa funktioner genom hela projektet.

2.2 Teamreflektioner

Vår generella uppfattning är att ingen av oss hade klarat av denna typ av projekt på egen hand, med tanke på den tidspress som förekommit. För att kunna använda PalPointer är det dessutom en förutsättning att minst två personer använder applikationen samtidigt. Ett gruppsamarbete har med andra ord upplevts som en nödvändighet. Att jobba i team

om fyra istället för att jobba själv innebär däremot inte att arbetet går fyra gånger snabbare, eftersom det är omöjligt att ha exakt koll på vad man ska göra själv och vad andra har gjort utan att lägga någon tid på koordinering. Detta gör att fungerande kommunikation är en förutsättning, vilket konstaterades redan i reflektionerna av scrum och XP. Sen projektet började på allvar i början maj har kommunikationen mellan gruppmedlemmarna fungerat väldigt bra. Detta beror främst på att samtliga prioriterat projektet och majoriteten av tiden suttit i samma rum och kodat. Bra kommunikation och gediget, målmedvetet arbete är det som gett resultat, och fungerat bra. Något som varit sämre är att det inte är någon som haft ett övergripande ansvar och strukturerat upp arbetet, vilket ibland har lett till lite meningsskiljaktigheter omkring vissa beslut kring applikationens utformning. Om det funnits någon som sett till att fler user stories definierats och mer iklätt sig rollen som kund hade kanske beslutsfattandet underlättats. Dessa diskussioner har dock varit förvånande få och arbetet har flutit på över förväntan.

Utöver reflektionerna ovan har vi även upplevt samarbetet inom gruppen som en extra kick och ökad arbetsmotivation, detta på grund av gemensamma framgångar och möjligheten att glädjas tillsammans.

2.3 Gruppsammansättning

Vid projektets början hade samtliga gruppmedlemmar i princip identisk programmeringsbakgrund och väldigt lika erfarenheter av att skriva kod. Vi går alla industriell ekonomi med teknikbas IT, och ingen av oss har direkt programmerat nämnvärt utanför skolan. Detta medförde både fördelar och nackdelar. Den främsta fördelen med att vara en så homogen grupp har varit att alla varit delaktiga och involverade i princip hela projektet, en ytterligare anledning till detta är att vi upplevt oss ha samma ambitionsnivå. Vi har haft samma målsättning och samma syn på hur och hur mycket vi velat arbeta, vilket har gjort att vi helt klarat oss undan dispyter. Nackdelar med detta är dock att om det funnits ytterligare gruppmedlemmar som haft mer spetskompetens inom mjukvaruutveckling, så hade vi kunnat lägga mer fokus på dokumentation och projektledning och då kunnat prioritera andra delar utöver själva kodandet mer. Det hade förmodligen gjort att vi fått en bättre bild över hur utveckling efter agila metoder verkligen fungerar, förmodligen hade det även gett ett bättre slutresultat, men vi tvivlar starkt på att samtliga gruppmedlemmar då hade utvecklat sina programmeringsskickligheter i lika stor utsträckning. Att tvingas bredda sig, och lära sig nytt istället för att göra det man redan kan, är nyttigt och samtliga medlemmar är nöjda med gruppens resultat. Med tanke på projektets intensitet på slutet underlättade det avsevärt att vi var en homogen grupp där alla hade samma målsättning. Vid ett mer avancerat mjukvaruutvecklingsprojekt som pågår under längre tid hade det däremot antagligen varit bättre att forma en grupp med olika kompetenser där medlemmarna kompletterade varandra på ett annat sätt.

3 Reflektion kursupplägg

Software engineering project har enligt samtliga gruppmedlemmar varit en av de roligaste kurserna som vi läst under våra tre år på Chalmers. Mycket beror på att kursen haft ett helt annat upplägg än övriga kurser och anpassats efter hur det ser ut i verkligheten. Kursen har innehållit lärorika föreläsningar, samtidigt som det ändå varit

mycket fokus på projektet. Det har varit bra med handledning en gång i veckan, och möjlighet till ännu mer handledning i slutskedet av kursen.

Det som varit sämre var den höga tröskeln innan vi på allvar kunde komma igång och programmera. Enligt examinator har det även tidigare år varit få grupper som kommit igång innan påsklovet. I år låg påsk dock väldigt sent, och när påsken väl var över var vi precis inne i sista veckan av vårt kandidatuppsatsskrivande, vilket gjorde att vi på allvar inte började förrän ännu en vecka senare.

3.1 GitHub

En aspekt som gjorde att vi inte kom i gång tidigare var versionshanteringsprogrammet GitHub. Sedan vi fick i gång det har det dock fungerat mycket bra, men det var mycket som strulade innan dess. Uppskattningsvis la två av gruppmedlemmarna nästan en halv normal arbetsvecka på att få det att fungera, och fick trots detta be om hjälp från en annan klasskamrat flera gånger. Det hade därför sparat många timmars ångest om det funnits en video eller någon annan slags tutorial för hur man installerar GitHub både på mac och pc. Detta är alltså ett hett tips att införa tills nästa år, sannolikt skulle det även göra att handledarna slapp att lägga värdefull tid på GitHub-problem.

GitHub är dock som sagt något som fungerat bra, när det väl var installerat och fungerade på samtliga gruppmedlemmars datorer. Vi har arbetat med ett stort antal brancher, och gjort en ny så fort vi börjat med någon ny funktionalitet. När denna ansetts fungerande och testats har koden integrerats i vår developer-branch. För det mesta har detta gått bra, men ibland har det uppstått mindre problem, som lett till lite krångel och att koden inte fungerat helt som den ska efter en sammanslagning av två brancher. Detta har dock varit relativt lätt att korrigera, men ibland krävts lite merarbete för att få bort felen i Eclipse. Dessutom upplever vi det som att vi blivit bättre på att undvika denna typ av fel allt efter kursens gång, bland annat genom ökad förståelse för vilka typer av förändringar som är lämpliga att utföra i master- kontra underliggande brancher. Det har därför varit flest fördelar med versionshantering på detta sätt, då det gjort att gruppmedlemmar kunnat arbeta samtidigt, utan att behöva oroa sig för problem i ett senare skede.

3.2 Föreläsningsschema

Något annat som skulle kunna övervägas hade varit att lägga fler föreläsningar i början av kursen. Då hade arbetsbördan blivit mer jämnt fördelad under hela läsperioden, och man hade också haft all relevant kunskap redan innan man börjat programmera på allvar. Som det såg ut i år la vi väldigt lite tid veckorna innan påsk, och sedan oerhört mycket efter påsk. Om alla föreläsningar hade legat innan påsk hade arbetsbördan jämnats ut avsevärt.

4 Reflektion testning

Under hela projektet har vi ägnat oss åt testning, vilket har lett till att arbetet gått stadigt framåt. Däremot är det inte förrän i slutet av projektet som vi använt oss av en utstuderad metodik för testningen. De typer av test som vi utfört är framför allt acceptanstest och JUnit-test. Eftersom inte någon av oss hade en androidtelefon har vi lånat till oss extratelefoner av kompisar och föräldrar så vi haft ett antal telefoner att tillgå. I synnerhet eftersom vår applikation bygger på gps-funktionen var det nödvändigt att testa med telefoner och inte enbart med emulatorn på datorn. För att säkerställa att gps:en fungerar som den ska och pilen pekar mot utsatt mål har vi ägnat många timmar och gått omkring med våra PalPointers i gång utanför skolbyggnaderna.

Varje gång innan vi har integrerat ny kod med äldre kod i vår developer-bransch i GitHub har vi varit noga med att testa och säkerställa att den varit funktionell. Det hade antagligen underlättat om vi redan från början hade bedrivit en mer systematisk testning, och hela tiden utfört acceptanstest i samband med att koden integrerats i GitHub. Nu var det först när i princip hela applikationen var färdig som vi antecknade våra acceptanstest och gjorde dem systematiskt. Det hade definitivt blivit mindre jobb om vi gjort denna mer strukturerade typ av blackbox-testning hela tiden, då vi sluppit att göra om alla våra test på slutet. Samtidigt gjorde vårt sätt, att testa samtliga krav i slutet, att vi fick ett sista test av hela appen. Detta gjorde att vi kunde upptäcka en del små oklarheter som vi inte tänkt på tidigare. Så även om det gav merjobb, tror vi att det ledde till en bättre slutprodukt.

Det vi gjorde under testningen var främst att validera applikationen, vilket innebär att vi jämförde med vision och krav och såg till att vi implementerat det som vi skulle i appen. Eftersom vi utvecklat efter agil metodik, och inte vattenfallsmetoden, var det lite svårare att verifiera appen. Detta eftersom det inte fanns någon detaljerad specifikation att jämföra med, utan endast user stories som beskrev funktionaliteten, och inte specificerade detaljer om hur den skulle implementeras.

Förutom acceptanstesten har vi som sagt även gjort några JUnit-test, som kan beskrivas som en typ av white-box-testning. Eftersom få av våra metoder returnerade något har vi valt att göra acceptanstest som nästintill täcker samtliga av applikationens krav, och endast komplettera med några JUnit-test där detta varit möjligt. För att försöka öka korrektheten och trovärdigheten av JUnit-testen har dessa utförts av personer som inte varit ansvariga för utvecklingen av den aktuella kod-sekvensen, eftersom denna person då inte har kännedom om några eventuella kryphål. Även JUnit-testen gjordes i slutet av utvecklingsprocessen. Dessa test hade varit ännu mer lämpligt att göra kontinuerligt, i samband med när koden skrivs. Om vi hela tiden arbetat mer testdrivet hade vi kunnat skriva koden på ett sätt så den kunnat testas bättre. Nu hade vi inte detta som vårt huvudsakliga fokus, utan jobbade främst med att skriva funktionell kod. I liknande kommande projekt skulle vi definitivt välja att arbeta lite mer testdrivet och göra white-box-testerna mer kontinuerligt, och förmodligen även ha med kontinuitet i acceptanstestningen, även om vi upplevt vissa fördelar med att systematiskt utföra och dokumentera tester i projektets slutskede.

En ytterligare form av testning hade kunnat vara att testa den funktionalitet som från början finns inbyggd i telefonen och som vi valt att använda oss av. Detta hade varit väldigt lämpligt för vårt projekt, men dessvärre insåg vi inte det förrän i efterhand. Ett exempel på detta är magnetkompassen. Denna kompass är bristfällig, med tanke på missvisning och deviation. Ett alternativ till att använda sig av magnetkompassen hade kunnat vara att använda sig av “backtracking” av gps-positioner, vilket innebär att

personens riktning beräknas utifrån den nuvarande positionen och en tidigare position. Detta tillvägagångssätt hade eventuellt gett en mer exakt återgivning av riktningen (så länge gps-positionen är korrekt), men en sämre uppdatering och rörelse av pilen. Denna typ av reflektioner borde ha hanterats och utvärderats i ett tidigt skede av projektet, efter det att telefonens inbyggda funktioner hade testats.

5 Reflektion icke-processpecifika beslut

Utöver traditionell programmering för android i Eclipse har vi använt oss av två olika sorters databaser, Microsoft Azure och SQLite. Eftersom vår applikation kräver att två telefoner utbyter information ansåg vi att en databas online, som flera telefoner kan komma åt via datatrafik, var lämpligt. Valet föll då på Azure, eftersom den upplevdes som enkel och användarvänlig. Den tjänst som använts (Azure erbjuder ett flertal tjänster) är begränsad till att viss information som är förknippad med användaren, i detta fall telefonnummer och position, kan lagras. För PalPointer, så som den ser ut idag, är detta ett bra alternativ. Om ytterligare funktionalitet ska integreras i applikationen i ett senare skede finns det en risk i att den typ av tjänst som i dagsläget används är allt för begränsad.

På ytterligare ett ställe använder vi en databas, och det är för att lagra vänners kontaktinformation, så man slipper skriva in denna flera gånger. Här har vi valt att använda oss av SQLite, detta eftersom denna kontaktfil är individuell för varje användare och ingen annan behöver ha tillgång till den. Genom att inte använda den kontaktlista som användaren har i telefonen behövs ingen tillåtelse för appen att få åtkomst till den. Detta ser vi som positivt av integritetsskäl. När filen bara behöver sparas lokalt på telefonen fanns ingen anledning att använda sig av någon mer avancerad lösning än SQLite.

Utöver dessa reflektioner har vi även insett att det antagligen hade varit bättre om vi hade använt oss av telefonens unika id vid inloggning, istället för Facebook-inloggningen. Anledningen till detta är att vi nu varit tvungna att hantera diverse fel som kan uppstå under själva inloggningen mot Facebook.