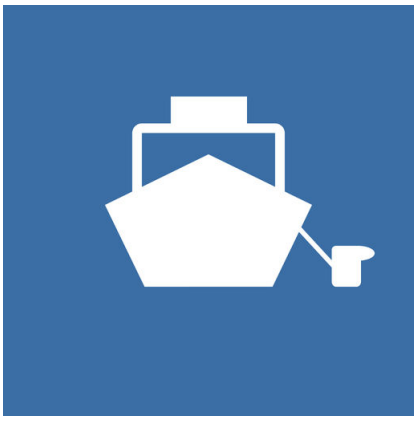


Reflektionsrapport (Group Reflection)

DAT255 - Software Engineering Project
VT 2018

JULIA BURMAN
JAKOB DALENBÄCK
ERIK GEDDA
PHILIP GÖRANSSON
AGNES KARLSSON
OSCAR KORSHAVN
JESSICA KÄRRBERG
ANTON LINDEGREN

DAT255 Software Engineering Project
Chalmers Tekniska Högskola
Göteborg, Sverige 2018

Innehållsförteckning

1
Introduktion

Under nio läsveckor har vi arbetat med att vidareutveckla applikationen Portable CDM. RISE gav oss som uppgift att anpassa applikationen efter Terminal 2 och deras behov. I vårt arbete representerades Terminal 2 av Preem med kontaktperson Henrik Sahlberg. Vår version av appen innehåller nu flertalet lösningar som vi anser ska underlätta arbetet för Teminal 2.

I arbetet med att utveckla appen har scrum-metodiken varit central. Det agila arbetssättet i kombination med ett nytt programmeringsspråk har varit både utmanande och utvecklande. I denna rapport följer reflektioner på arbetet i vilka vi tar upp problem och lösningar som uppstått under arbetets gång. Vi ger också vår framtidsvision där vi beskriver vad vi tar med oss till framtida projekt, hur vi vill utvecklas inom området och vad som krävs för att nå dit.

2
Projektets utformning

I detta avsnitt görs reflektioner kring projektets utformning och hur dess arbete förändrats under tiden som gått.

2.1 Scope

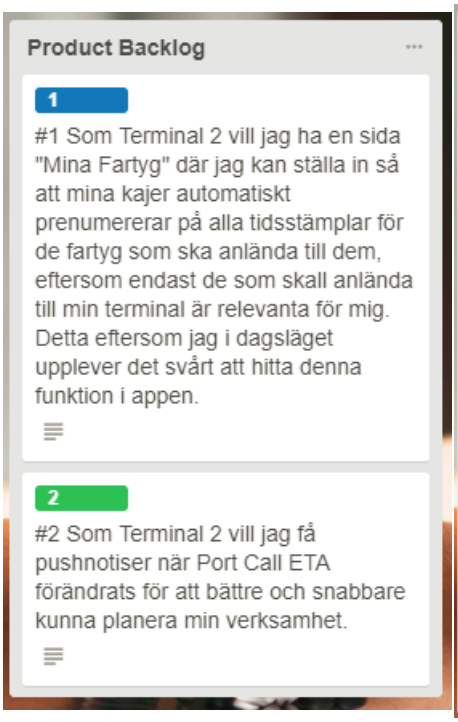
The chosen scope of the application under development including priority of features and for whom you are creating value.

Omfattningen av applikationen har under arbetets gång ändrats ett flertal gånger. Detta har främst berott på svårigheter med att utföra programmeringen och därmed även att uppskatta velocityn på user stories.

I början av arbetet trodde vi nämligen att vi skulle ha möjlighet att utföra stora delar utav det som Terminal 2 efterfrågade (se figur 2.1 nedan). Under arbetets gång växte sig dock utmaningarna i att programmera i ett helt nytt språk för stora och scopet minskades därför till att endast omfatta user story 1 och 2 (se figur 2.2 nedan). Samtidigt som vi bestämde oss för att minska scopet gjorde vi också om estimeringen av våra user stories, vilket paradoxalt nog resulterade i en scope-creep. Detta eftersom vi vid detta lag uppfattade att user story 1 och 2 skulle ta mycket längre tid än vad vi tidigare trott. När vi sedan lärde oss att vi inte skulle uppskatta velocity i timmar (mer om detta under *4.1 User stories*) minskade dock scopet på pappret igen, men scopet var ju detsamma i verkligheten: Fortfarande User story 1 och 2.

Figur 2.1: Product Backlog v.1

User story 1 innebar att skapa en ny vy där man endast kunde se de fartyg som skulle till en specifik hamn och user story 2 innebar att ta fram pushnotiser som varnade när fartyg hade uppdaterade tidsstämplar. Att skapa pushnotiser visade sig dock innebära flertalet svårigheter, och för att fortfarande kunna tillföra värde till kunden undersökte vi därför vad en minimal viable product skulle innehålla genom att fråga Preem vad de framförallt ville veta med pushnotisen. Därefter ändrade vi user story 2 till att istället för pushnotiser endast markera de fartyg som hade uppdateringar, vilket var det Preem värdesatte, med hjälp av en symbol (se figur 2.3 nedan).



Figur 2.2: Product Backlog v.2

Figur 2.3: Product Backlog v.3

Under de tre sista sprinterna jobbade gruppen på med user story 1 och 2, men när deadline på projektet närmade sig fanns det en bugg i lösningen för user story 1 som vi hade svårt att åtgärda. Detta ledde till att user story 1 förändrades och vi la även till user story 3, som omformulerats från första veckan (se 2.1 ovan). User story 1 och 3 innebar att flera utav de förändringar som var tänkta att vara i den vy som vi skulle skapat istället lades till på startsidan(se figur 2.4 nedan).

Figur 2.4: Product Backlog v.4

Anledningen till att vi behövde minska omfattningen av projektet flertalet gånger tror vi som sagt främst beror på att vi överskattade vår förmåga inom JavaScript och underskattade utmaningarna att förstå Redux. Överlag var uppskattningen av arbetsbelastningen svår, och omfattningen av våra user stories svåra att estimera. På grund av svårigheterna prioriterades andra saker och bestämmandet av omfattningen försvann ibland i planeringen av nästkommande sprint.

I framtida projekt vill vi arbeta med ett mer realistisk scope. För att göra detta bör vi börja i mindre skala från början och istället skala upp projektet vid behov. Detta tror vi kommer göra att vi kan göra färre saker ordentligt, istället för flera saker med lägre värde i slutändan. Att öva på att uppskatta arbetsbördan och velocityn för olika stories är ett annat sätt att skapa ett mer realistiskt scope för projektet. Genom att öva på detta blir vi bättre på att känna oss själva och vår egen förmåga, vilket gör att vi tillsammans med produktägaren kan sätta ett scope som faktiskt kan realiseras.

För att åstadkomma ett mer realistiskt scope att arbeta efter i nästkommande projekt är det viktigt att vi tar oss mer tid att uppskatta velocityn, även om detta är svårt till en början. För att komma igång med det skulle vi, till exempel, kunna utföra övningen som vi gjort på en av föreläsningarna, där alla grupper fick uppskatta velocityn av en user story. Således skapades ett spann för var velocityn bör ligga och den blir då enklare att uppskatta.

Ett annat sätt att åstadkomma ett mer realistiskt scope i nästkommande projekt är att jobba tydligare med prioriteringar under planeringsfasen av sprintarna. Exempelvis prioritera momenten som innebär att estimera velocityn, även om det upplevs som att själva programmeringen är av högsta prioritet för stunden. Om vi kan få en tydligare prioritering bland tasksen och inte bara user storiesen skulle arbetet kunna effektiviseras ytterligare, vilket i sin tur gjort att gruppens totala arbetsinsats kunnat öka. Därmed skulle scopet kunna öka i nästkommande projekt, utan att arbetsbördan nödvändigtvis blivit tyngre.

2.2 Social contract

Your social contract, which means you should create one in the first week.

Redan första läsveckan initierades, som en del av de riktlinjer som fanns i kursen, ett socialt kontrakt. Detta kontrakt beskrev olika förhållningssätt och förväntningar över hur arbetet skulle gå till inom gruppen. Samtliga medlemmar skrev under.

Under arbetets gång användes en del av riktlinjerna till stor grad, såsom hur möten ska hållas. Andra delar, såsom hur beslut skulle tas om gruppen var oense, behövde aldrig implementeras.

Gruppens arbete har överlag varit effektivt och inga större konflikter har uppstått. Det största hindret var inte internt utan kom från svårigheterna i att installera och starta upp utvecklingsmiljön, vilket ledde till en del frustation. De problem som har funnits inom gruppen är att kommunikationen i vissa fall har varit bristfällig vilket har påverkat koordineringen av arbetet. Exempelvis var det flera gånger gruppen hade olika uppfattningar om när mötena skulle hållas i och med att mötestiden annonserades sent på kvällen.

I framtida projekt tar gruppen med sig erfarenheten i att lägga mer vikt på hur kommunikationen ska skötas och planera in möten i förväg. Att ha inplanerade stående möten varje vecka visade sig vara väldigt viktigt och underlättade arbetet från vecka 1. Över lag fungerade gruppkontraktet bra och satte upp tydliga mål och riktlinjer för arbetet.

För att åstadkomma en bättre kommunikation i gruppen under nästkommande projekt är det viktigt att tala om hur gruppmedlemmarna upplever att kommunikationen sköts. Det kan vara bra att ha ett utvärderingsmöte av gruppkontraktet en bit in i projektet för att ta upp denna typ av frågor. Det skulle även kunna vara en stående punkt under sprint retrospektiven, eftersom det handlar om hur gruppen arbetar tillsammans.

2.3 Success criteria

The success criteria for the team in terms of what you want to achieve with your application.

I början av arbetet hade vi inga tydligt definerade success criterias. Vårt success critera var ”en bra slutprodukt vi är nöjda över, en nöjd kund, samt ett bra resultat på kursen för samtliga gruppmedlemmar”. Vi insåg att vårt success criteria inte var tillräckligt definierat men eftersom vi varken hade fått igång utvecklingsmiljön eller förstått programmeringsspråket ansåg vi då att det var svårt att få grepp om vad som var möjligt att utföra och därmed vad vi ville lyckas med.

Vårt ursprungliga vaga success critera kvarstod under flera sprinter, vilket ledde till ett spretigt arbete där vi hade olika uppfattning om vad vårt success criteria verkligen innebar. Med tiden minskades dock osäkerheten i projektet och vi fick en allt bättre förståelse för JavaScript, utvecklingsmiljön och hur vi kunde möta produktägarnas förväntningar.

Effekten av vår förbättrade förståelse var att vi kunde åtgärda vår tidigare spretiga målbild och definera tydligare success criterias. Under denna period gjorde vi ytterst små framsteg i koden. För att öka motivationen i gruppen satte vi därför upp success criterias för varje vecka, vilka exempelvis kunde vara: ”*Vi vill avklara den andra user storyn och bryta ner en ny user story till tasks som vi kan börja jobba med under nästkommande sprint*”. Till skillnad från tasks kan dessa success criterias behandla för kunden icke-värdeadderande uppgifter eller andra saker som vi anser är ett mått på success, såsom stressnivåer eller nedbrytning av tasks. Dessa veckovisa success criterias syftade till att bryta ner vad vårt tidigare övergripande success criteria skulle innebära för kommande sprint. På så sätt kunde vi på ett strukturerat sätt jobba mot vårt långsiktiga mål och samtidigt öka motivationen i gruppen. De kortsiktiga målen gav därmed en bra bild av att vi rörde oss mot vårt långsiktiga mål. Vi hade dock inte sett över hur mycket närmare varje veckovist success criteria tog oss mot målet, vilket gjorde att vi inte hade full koll på våra framsteg.

I kommande projekt anser vi att det är fördelaktigt att fortsätta med veckovisa och långsiktiga success criterias. Vi skulle dock vilja komma igång med detta tidigare för att undvika den period utav spretigt och utstrukurerat arbete som vi uppfattade denna gång. Detta tror vi kommer leda till en bättre koordineringen av gruppens arbete då alla strävar efter samma mål, samt att ha tydligare delmål som håller motivationen uppe. Genom att få tydligare delmål är det även enklare att se syftet i de tasks som utförs under sprinten, eftersom de bidrar till att vi tar oss närmare våra övergripande success criterias.

För att säkerställa att våra veckovisa success criterias leder till vår långsiktiga plan vill vi i början av arbetet sätta upp en plan för dessa med tydliga målstolpar. Om inte gruppen gemensamt vill ta ansvaret för en gemensam planering kan det vara bra att införa en ”successansvarig” som ser till att en plan med delmål upprättas

samt att målen arbetas mot. Detta sammanfaller till viss mån med KPI:n burn up chart.

2.4 KPI

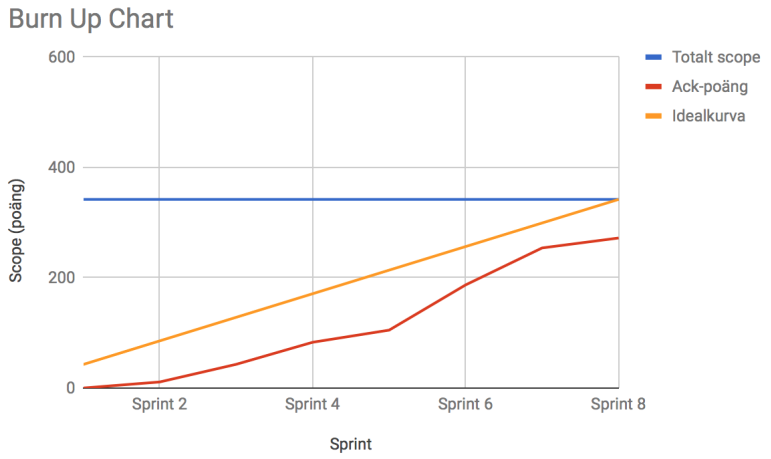
The three KPIs you use for monitoring your progress.

I början av projektet valde vi stressnivå i gruppen, defekter i koden samt burn-up chart som KPI:er. Burn-up chartet användes för att visa projektets scope och hur mycket av det som utförts över tid. Eftersom vi inte hade kommit igång med vår kod kunde vi inte använda defects direkt. Vi hade också svårt att applicera burn up chartet innan vi lyckats dela upp våra tasks.

Stressenkäter skickades ut kontinuerligt för att mäta stressnivåerna, men i takt med att vi tappade fokus på scrum-arbetet en bit in i projektet föll även tanken om att följa upp resultaten från dessa enkäter bort. Vi insåg också att defekter inte var den mest givande KPI:n eftersom vi alla satt och programmerade i våra respektive developer-branscher under stora delar av projektet, och där växlade antalet defekter kraftigt. Defekter som KPI hade kanske gett oss mer om vi tidigare hade börjat mergea olika branscher till masterbranschen, men nu skedde ingen sådan merge förrän i slutet av projektet. Anledningen till att vi inte mergeade oftare och då mätte defekterna är troligtvis att vi hade dålig kunskap om programmeringsspråket och inte tidigare använt oss av en trädliknande utvecklingsstruktur på koden. Därför använde vi oss inte alls av detta KPI under projektet.

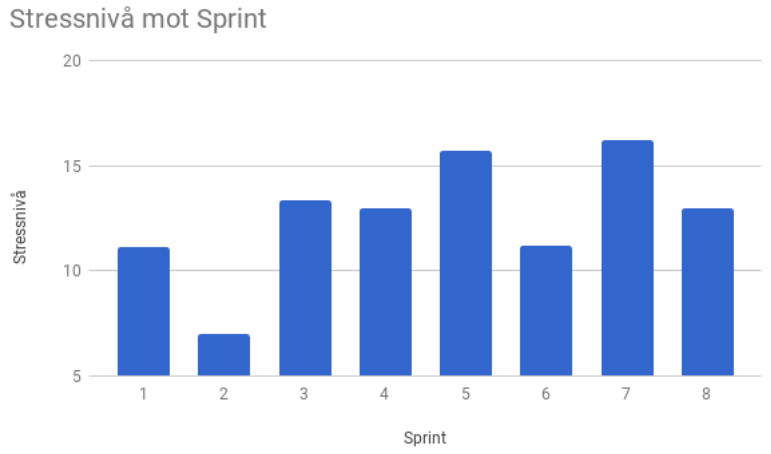
När det gäller vårt burn-up chart dröjde det innan vi fick igång det, helt enkelt eftersom det dröjde innan vi lyckades komma igång med koden och dela upp tasks. Vi använde här Google Spreadsheet som verktyg för att föra in data kring scope och framfart. Som beskrivs i avsnittet User Stories började vi dock uppskatta user stories och tasks i timmar, vilket ju inte var optimalt. Eftersom vi inte heller var så konsekventa med hur vi satte dessa gav inte burn-up chartet speciellt mycket och vi tappade därför fokus från det.

Fram mot slutet av projektet tog vi tag i scrum igen och därmed också våra KPI:er. Efter att ha strukturerat om hur vi poängsatte tasks och stories kunde vi nu få en bättre bedömning av vår velocity. Istället för timmar använde vi oss nu av arbetsbörda. Detta gjorde att vi de sista sprintarna kunde använda ett fungerande burn-up chart och återkonstruera de veckor vi varit dåliga på att följa upp våra KPI:er, vilken visualiseras i figur 2.5.



Figur 2.5: Visar gruppens burn-up chart. Chartet visar att det fanns en viss startsträcka, och att produktiviteten var som bäst under sprint 6 och 7. Under sprint 8 insåg vi att vi inte skulle hinna med en av våra tasks och gjorde därför en enklare task istället.

Stressnivåerna fortsattes samlas in och sammanställdes för att resultatet skulle gå att ta del av. Detta skedde dock först efter den värsta stressen var över enligt resultatet, vilket kanske tog bort lite av syftet med att mäta detta KPI. En överblick över stressnivåerna i gruppen finns i figur 2.6.



Figur 2.6: Visar gruppens stressnivåer under samtliga sprintar. Stressnivåerna har fluktuerat en del under arbetets gång och har påverkats av yttre faktorer som till exempel inlämnandet av kandidatarbete.

Till framtida projekt hade det varit användbart att använda sig av KPI:er under hela projektens gång och följa upp dessa kontinuerligt. Exempelvis kan det nog vara bra att kontinuerligt följa upp ett burn-up chart för att få en bättre överblick över hur gruppen ligger till. Idealt sett bör detta chart vara lättillgängligt för alla i gruppen för att sprida information om gruppens prestation. Exempelvis skulle detta

kunna ligga i Trello under fliken ”Notes” eller om en ny flik ”KPI” skapas. Stressenkäterna skulle också tagits upp under arbetets gång, minst en gång i veckan, för att eventuellt kunna åtgärda det som stressar gruppmedlemmarna. Defekter i koden är kanske inte ett KPI som behöver tas med till framtida projekt, om inte det kommande projektet är lättare att programmera eller innehåller fler merges.

För att lyckas med KPI:erna i framtida projekt är det viktigt att de används från början så att trender kan hittas och de kan utvecklas under tid. KPI:er bör exempelvis vara en stående punkt på retrospective-mötet och KPI-gruppen bör kontinuerligt redovisa hur gruppen ligger till utifrån de olika KPI:erna. Det skulle därmed förtydligas att det inte bara åligger KPI-ansvariga att samla in information om KPI:erna, utan även att kontinuerligt redovisa dessa för gruppen under varje sprint. Från detta projekt tar vi med oss den vetskapen. Vi bör också kunna ta med oss många av de felsteg vi gjorde med de KPI:er vi faktiskt använde, för att inte upprepa dem framöver.

2.5 Relation till litteraturen

Relation to literature and guest lectures (how do your reflections relate to what others have to say?).

I början av kursen lästes mycket litteratur. Den litteratur som vi främst relaterat till har varit litteratur om det agila arbetssättet. I ett tidigt stadie fanns det svårigheter i att förstå de problem som de agila metoderna skulle lösa, vilket berodde på att vi aldrig tidigare har arbetat med mjukvaruutveckling i stor grupp och därmed inte stött på dessa problem.

Trots att vi tog med oss kunskaper om agil utveckling från föreläsningarna i början av kursen hade vi ändå svårt att applicera de agila metoderna korrekt. Vi började stöta på många av de problem som beskrevs i litteraturen och som vi från början försökt undvika. I takt med att problemen uppstod ökade dock förståelsen för det vi läst i litteraturen och vi började tillämpa metoderna korrekt. Detta kunde till exempel ses när vi till en början delade upp våra user stories horisontellt. När vi stötte på problem på grund utav detta hade vi lättare att ta till oss artikeln ”Splitting user stories - the hamburger method” för att på ett bättre dela upp våra user stories.

En av de viktigaste punkterna vi tog med oss från föreläsningarna var tipset från gästföreläsningen med Michael Öhman från Trine, som föreläste om agila arbetsmetoder. Han menade på att om man vill lära sig ett nytt programmeringsspråk så är det enklast att börja koda direkt och göra inkrementella förändringar. Detta har vi lyckats applicera i projektet och anser ha hjälpt oss tackla många av de svårigheter vi haft med kodandet.

I framtida projekt önskar vi kunna tillämpa metoderna som vi fått med oss från kursen direkt, istället för att först göra de fel litteraturen beskriver och sedan lita

på vad litteraturen säger om misstagen. Vi tror att de erfarenheter vi fått från kursen skapat den förståelse som krävs för att kunna tillämpa de agila verktygen, vilket gör att vi kommer tillämpa verktygen direkt istället för när problem väl uppstår. Därmed vill vi i framtida projekt sträva efter att arbeta mer aktivt med att ta till oss litteratur, så att arbetet kan effektiviseras.

För att tillämpa det litteraturen säger snabbare är det viktigt att vi vågar inse våra misstag och ändra vårt arbetssätt. För att åstadkomma detta är den erfarenhet vi samlat på oss under projektets gång viktig, men det bör även klargöras tidigt i arbetsprocessen att vi ska lägga större tyngd vid litteraturen och vad den har att säga. Om vi ska arbeta mer aktivt med litteraturen bör det även ingå som en naturlig del i vårt arbetssätt att studera litteraturen. Exempelvis skulle detta kunna inkluderas som en del i det sociala kontraktet, där vi klargör att detta är det arbetssätt vi vill använda i projektet. I framtiden kommer vi dock med största sannolikhet fortsätta stöta på likande problem och det är därför viktigt att lyfta fram dessa under arbetsprocessen, så att vi kan fortsätta lära utav dessa och bygga på vår erfarenhet. När vi nu ser tillbaka till projektets gång så kan finns det ett värde i att vi faktiskt stötte på många av de fallgroparna som litteraturen tog upp. Det är en svår process att driva ett utvecklingsprojekt och varje misstag vi gjorde tar vi med oss till framtida projekt

3
Programmeringsmetodik

I detta kapitlet diskuteras de ämnen som berör programmeringsmetodik och applikationens utformning.

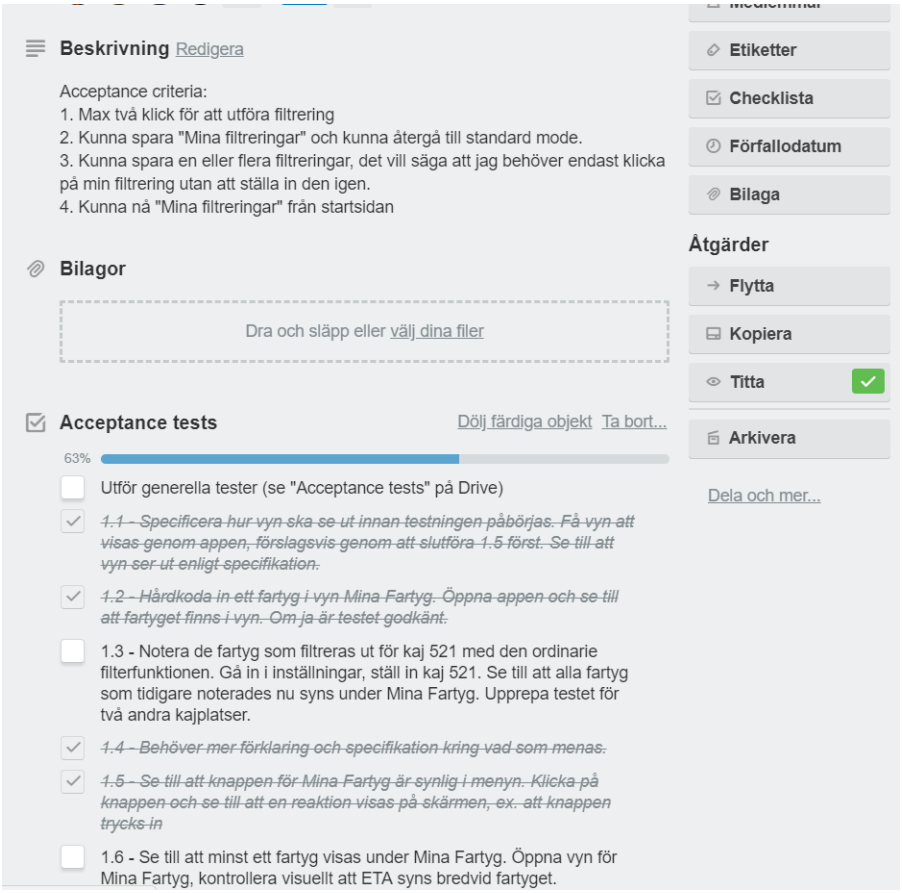
3.1 Acceptance criteria

Your acceptance tests, such as how they were performed and with whom.

Till en början utformade vi inga acceptanskriterier. Den största anledningen till detta var att vi kände en stor stress över att komma igång med kodningen och utvecklingsmiljön, vilket gjorde att vi slarvade med grundarbetet och våra user stories. Ytterligare en anledning var att vi inte helt förstod hur de skulle tas fram. Detta gjorde senare att vi fick svårigheter med att definiera när en story och task var klar, eller vad uppgiften egentligen ämnade åstadkomma.

En bit in i projektet insåg vi att vi måste utforma acceptanskriterier och passade då på att även utforma acceptanstester. Vi insåg då att det finns likheter mellan acceptanskriterier och acceptanstester och fick svårt att skilja dem åt. För att få klarhet i frågan sökte vi litteratur på internet samt diskuterade skillnaderna med Håkan Burden och Jan-Philipp Steghöfer. Vi kom fram till att eftersom vi redan delat upp våra user stories i tasks och givit dessa primära acceptanskriterier blir de starkt sammankopplade med våra acceptanstester. Detta var dock okej och kan förekomma, enligt Burden och Steghöfer. Testerna är till för att dels checka av att acceptanskriterierna är uppfyllda, men kan även inkludera testning av andra funktioner.

Efter att vi fått klargjort skillnaden på acceptanskriterier och acceptanstester förfinade vi de primära acceptanskriterierna vi gjort. Vi funderade själva över vad vi ansåg som rimligt minimum för varje task och checkade av med den ”riktiga” produktägaren från Preem, Henrik Sahlberg. Eftersom vi haft svårigheter med kodningen bestämde vi oss för att utforma testerna som praktiska tester där vi testade funktionaliteten genom blackbox tester. Därefter förde vi in kriterierna och testerna på våra kort i Trello, se bild 3.1. Utöver de primära testerna för varje enskild task utformades även generella tester som skulle utföras. Dessa tester innefattade bland annat att låta gruppmedlemmar använda de nya funktionerna och låta en utomstående part använda koden och ge input, se appendix A.



Figur 3.1: Bilden visar hur acceptanskriterier och acceptanstester fördes in i Trello.

I efterhand har vi förstått vikten av att använda tydliga acceptanskriterier och acceptanstester. Om vi hade fått en chans att göra om projektet skulle vi kommit igång tidigare med testerna och haft tydligare kriterier. Som figur 3.1 visar använde vi i slutet av projektet ett system där vi skrev upp testerna för varje task direkt på user storyn och bockade av dem successivt. Detta fungerade bra och var ett effektivt sätt att inkorporera testningen i processen. Detta gjorde att arbetet blev mer strukturerat och vi insåg att det finns stora fördelar med att ha denna specifikation från början.

Om vi i nästa projekt har bättre kunskap om kodspråket hade vi även velat utforma kodtester. På så vis hade det underlättat att hitta fel i koden och förstå var det potentiellt kan uppstå merge-errors. Olika tillvägagångssätt är då att skriva kodtesterna innan koden skrivs, alternativt att de skrivs av en annan person än den som skrivit koden. Detta för att undvika konfirmeringsbias.

För att se till att komma igång med kriterier och tester tidigare är det viktigt att nästa gång inkludera detta i den initiala planeringen av projektet. Således definieras tydligare vad som ska göras och det blir mer tydligt vad de olika user storiesen ämnar uppfylla. Ett annat sätt att se till att kriterierna blir definierade tidigt är att genomföra en övning för att få gruppmedlemmarna att fundera kring vad som är minimal viable product (MVP) i de olika storiesen och återkoppla detta med

produktägaren.

3.2 Design of application

The design of your application (choice of APIs, architecture patterns etc).

Eftersom att vi fick en färdig, fungerande app så ansåg vi det inte nödvändigt för oss att välja varken API eller architecture patterns. Däremot var det viktigt att vi anpassade oss efter den struktur som fanns eftersom vår version av appen skulle skapa värde för produktägarna Sandra, Mathias och Henrik. Delar av vår app skulle alltså i teorin kunna användas och integreras i den befintliga appen. Vi insåg att vi därför skulle behöva hitta de designmönster och API:er som redan använts i appen.

Då vi inte hade någon förkunskap inom varken React-Native eller Redux hade vi till en början problem med att identifiera den befintliga designen. Arbetet med att identifiera detta blev därför krävande för gruppen vilket gjorde att vi inte kunde utveckla eller förstå appen. Till en början försökte vi identifera designmönstren genom att enbart titta på hur koden var uppbyggd och sammankopplad, eftersom att det var så vi gjort i tidigare kurser som byggt på kodspråket Java. För gruppen kändes det till en början inte heller relevant att veta enligt vilka principer som appen var uppbyggd. Dock insåg gruppen snabbt att det skulle ta en väldigt lång tid att förstå koden enbart genom att klicka sig runt och leta kopplingar.

Istället letade vi efter alternativa sätt att lära oss och tog hjälp av appens utvecklare, Pontus. Pontus bad oss kolla upp begreppen React-Native och Redux. Därefter krävdes mycket research på nätet för att förstå vad dessa två begrepp innebär och hur de samverkade. React-Native ger oss de API:er som använts i koden och Redux är det designmönster som använts. Insikten om appens design gjorde att vi kunde förstå den bakomliggande strukturen i appen, vilket i sin tur gav oss möjlighet att börja utveckla appen enligt det befintliga mönstret.

I ett eventuellt framtida projekt som innefattar en redan utvecklad produkt måste vi vara snabbare med att anpassa oss till den befintliga koden som finns. För att göra detta bör den bakomliggande designen vara något som vi prioriterar att förstå, redan till en början. Önskvärt hade varit att från början ha en klar och tydlig struktur över appens uppbyggnad, designmönster och datastrukturer som alla i gruppen förstår. Således kan arbetet i gruppen effektiviseras och tid på att förstå uppbyggnaden kan minskas.

För att åstadkomma detta bör vi vara mer lyhörda och inte enbart kolla i koden utan ta den hjälp som finns tillgänglig, exempelvis genom att fråga de personer som tidigare varit involverade i utvecklingen av appen. I detta projekt skulle det innebära att ha större kontakt med Pontus, Sandra och Mathias. Vi bör även ha ett mer konkret tillvägagångssätt för att se till att vi verkligen kollar upp de osäkerheter vi har, exempelvis genom att skriva upp de frågor vi har på en to-do lista och bocka

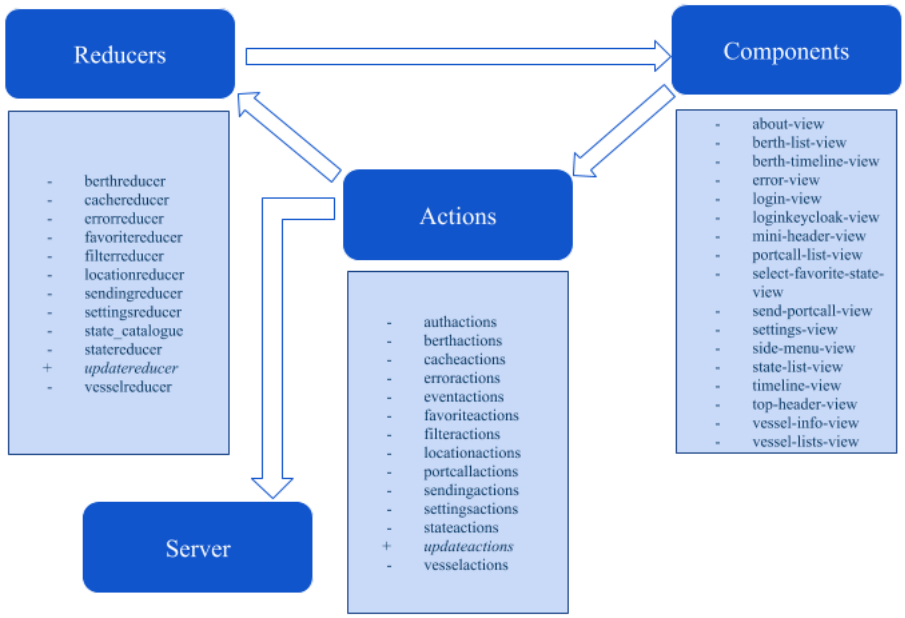
av dessa efter att vi pratat med relevanta personer.

Under projektet har vi även insett att designen påverkar utvecklingen avsevärt. Av denna anledning bör vi lägga stor vikt vid att välja design omsorgsfullt om vi jobbar i projekt där koden och dess struktur inte redan är utvecklad.

3.3 Strukturell översikt

The structural overview of your application (such as class diagrams, domain models or component diagrams).

Även i detta fall fanns det redan en struktur i appen som vi följde. När vi hade identifierat att det var både Redux och React-Native som användes lades mycket tid på att se på videos, söka på nätet efter guider och att klicka sig runt i koden för öka förståelsen för appens struktur. Detta ledde till att vi förstod att appen var uppbyggd av reducers, actions och components som alla samverkade. Vi anpassade oss efter denna struktur men kom att lägga till actions, components och reducers i den nya appen(se figur 3.2). Utöver detta ändrade vi inte struktren för appen.



Figur 3.2: Visar samspelet mellan reducers, components och actions för Redux. De listade objekten som har ett '+' framför namnet skapade av oss.

Då vi genom att ha förstått appens design kunde förstå strukturen i appen så anser vi att vi även här vill vara snabbare på att anpassa oss till den kod vi får i kommande projekt. Eftersom appens struktur till stor del bygger på apppens design blir

det därmed liknande målbild inför framtida projekt i denna fråga som i ovanstående rubrik. Vad som i framtida projekt är mer viktigt under denna punkt är dock att skapa en gemensam bild i gruppen över appens strukturella översikt och hur vi ämnar ändra den. Därmed sprids information om hur appen fungerar i sin nuvarande form bättre, vilket underlättar för gruppmedlemmarna i sin utvecklingsdel.

För att åstadkomma detta skulle vi i framtida projekt kunna ha ett dokument där varje medlem kan lägga till de eventuella funktioner som lagts till i applikationen, alternativt att detta tas upp under utvärderingen eller planeringen av respektive sprint. Detta är synnerligen viktigt för att undvika dubbelarbete, eller att liknande funktioner skapas av olika utvecklingspar i gruppen. På så vis skulle detta även bidra till en ökad effektivitet i projektet.

3.4 Översikt av beteende för applikationen

The behavioural overview of your application (for instance through use cases, interaction diagrams or similar).

Hur appen är tänkt att interagera med användaren är något vi arbetat med kontinuerligt under majoriteten av projektet. Mot slutet av projektet insåg gruppen att vi gärna vill visa våra idéer kring de user stories vi inte hann med i projektet. Därför valde vi att under sjätte, sjunde och åttonde sprinten utforma en visionsapp som stöd för att förmedla våra idéer. Denna kan ses genom att klicka här. I visionsappen är dels de funktioner som vi hunnit med att utveckla inkluderade, men även våra visioner om pushnotiser, ett enkelt sätt att se vilka övriga aktiviteter som fartyget planerat göra i hamnen, export till produktägarens ordinarie affärssystem (OAS) och en genväg till att filtrera PortCalls på location. Det vi faktiskt utvecklat är de olika snabbflikarna för olika stages ”under way”, ”planned” och ”berthed”, att en röd symbol syns bredvid det PortCall som ändrats sedan sist appen användes samt två sätt för att avmarkera symbolerna. Detta kan göras genom att använda knappen ”unmark all” eller genom att hålla inne på det PortCall man önskar avmarkera och sedan klicka på ”mark as read”.

I framtida projekt skulle en vision av denna typ kunna fungera som ett stöd för att visa gruppen målbilden och vad som behöver utvecklas. Det skulle även kunna agera som en hjälp i tester om kriteriet är att den utvecklade funktionen bör se ut som visionen. Vad vi skulle gjort annorlunda i ett potentiellt framtida projekt är att involvera produktägarna mer för att verifiera att våra idéer stämmer överens med vad de önskar att vi utvecklar.

För att åstadkomma detta i nästkommande projekt bör vi först bestämma om vi vill använda en visionsapp som målbild och i så fall ta fram denna i början av projektet. Således bör det planeras in som en del av förarbetet i projektet. Visionsappen skulle även kunna vara till hjälp för produktägaren att förstå hur gruppen ämnar utforma den fortsatta utvecklingen av appen och det hade kunnat vara en stående diskussionpunkt för att underlätta den kontinuerliga kommunikationen.

För att se till att produktägarna involveras mer bör vi till nästa gång se till att följa scrum-metodiken bättre, eftersom det ingår i sprint-reviewn att stämma av med produktägaren. På så vis ökas kontakten med produktägaren och det blir enklare att stämma av.

3.5 Kodkvalitetsvertyg

Code quality using a tool such as Findbugs.

Kodkvalitet var på agendan från projektets början då att detta var något som behandlades i de veckovisa reflektionerna och det togs upp på föreläsningarna. I och med att Android Studio är baserat på IntelliJ:s arkitektur så trodde vi initialt att vi kunde använda FindBugs tillägg direkt. Det visade sig strax att detta inte skulle fungera, då denna tanke uppkom innan vi insåg att vi inte skulle programmera i Android Studio. Samtidigt som vi insåg detta försökte vi hitta ett sätt att testa koden på i Atom. Detta gick inte att göra genom ett tillägg och vi hittade heller inget sätt att köra standalone testning. Vid detta laget var återigen stressen och frustrationen med att få igång appen och Expo så hög att testning lades på hyllan. Det fanns dessutom ingen kod att testa varpå vi sköt upp detta till ett senare skede, när vi faktiskt skulle ha lite kod att testa på. När vi väl började producera kod tänkte vi inte på att testa den eftersom mängden producerad kod var så liten. Vi hade, under den fasen i projektet, inga commits till masterbranchen eftersom alla dessa inkrementella förändringar låg i utvecklingsbrancher, som inte var tänkta att testa i. Vi förstod det som att utvecklingsbranchen alltid kommer ha defekter i sig och att det är hela idén med att ha en utvecklingsbranch. Eftersom branchen alltid är under utveckling är den därför orimlig att testa i som om det vore en slutprodukt.

I samråd med Burden insåg vi att vi behövde börja testa koden när vi började commita våra branches till master. När vi återigen inte hittade något sätt att formellt testa koden på blev vi lite oroliga, detta var ju trots allt något som vi skulle göra. Vi hade dock inte kunskapen att skriva JUnit tests eller få en testningsapplikation att fungera tillsammans med Atom. Trots att vi valde bort den formella testningen av kodens kvalitet under tiden vi utvecklade koden och trots att vi inte fick formella tester att fungera gjorde vi tester med hjälp av mindre formella metoder. Blackboxing var den dominanta metoden och säkerställde att kvaliteten på koden var okej med avseende på vad den faktiskt uträttade för användaren.

En annan metod som användes för att testa koden var att skriva ut objektattribut i konsolen på ställen vi tyckte var lämpliga eller där det någon gång tycktes uppstå fel under kodningen. Exempelvis där objekt var *undefined* och där liknande fel verkade ha sitt ursprung. Med tanke på att vi inte kunde språket eller arkitekturen särskilt bra var det mycket klippande och klistrande samt återanvändning av befintlig kod, vilket åtminstone borde minska risken för grova kvalitetsfel som redan inte fanns i koden.

I ett potentiellt framtida projekt kan det vara fördelaktigt att reda ut hur testningen

ska ske tidigare under projektet. Således kan tid sparas in på undersökande aktiviteter som, liksom i detta fall, inte tillför något värde till projektet. Arbetet kan troligtvis effektiviseras genom att integrera testningen av koden som en naturlig del av processen, exempelvis varje gång vi mergar en ny branch eller anser att en del kod är färdigskriven.

För att åstadkomma detta kan exempelvis hjälp tas av mer kunniga personer, som då potentiellt kan bidra med tips om relevanta testverktyg. För att se till att testningen av kodkvalitet blir en mer integrerad del av arbetet bör det inkluderas i definition of done (DoD). Därefter bör DoD användas som en tydlig kravbild, vilket vi varit dåliga på under projektet.

4
Scrum

I detta kapitel redogörs för hur scrum-metodiken har applicerats och använts i projektet, samt vilka verktyg som använts.

4.1 User Stories

Your user stories in terms of using a standard pattern, acceptance criteria, task breakdown and effort estimation.

Vi förstod tidigt att user stories var en essentiell del av scrum-metodiken. Eftersom det är grunden till allt vidare arbete började vi med att upprätta några olika user stories efter ett produktivt första möte med Terminal 2 - Preem. Detta gjorde vi direkt i vår Trelloboard, vilket var smidigt. Något som inte visade sig vara lika smidigt var att vi inte lade ner tillräckligt mycket tid på att definiera vad dessa user stories konkret skulle innebära för oss eller på vilket sätt de skulle hjälpa oss i vårt arbete. Till att börja med delade vi inte upp user stories i nog många och nog små tasks. Vi missförstod också hur vi skulle värdera och poängsätta de få tasks vi hade, då vi använde uppskattat antal timmar som skulle krävas för att göra klart tasken som mått. I vår stress och frustration med att få igång utvecklingsmiljön tänkte vi, i början av projektet, dessutom inte på att sätta upp någon tydlig definition of done och acceptance criteria.

Detta oavslutade arbete med user stories skulle komma att bli en av de viktigaste lärdomarna från kursen. Vi förstod efter ett tag att vi hade underskattat vikten av en stabil grund och värdet av ett gediget arbete med user stories. Det var i detta senare skede vi insåg att vi delat upp user stories i för få tasks och att detta ledde till att vi inte kunde jobba tillräckligt inkrementellt. Tasks var helt enkelt för stora för sprint-backlogen vid detta lag. När vi väl delade upp user stories i fler och mindre tasks blev det intressant nog lättare att fylla upp sprint backlogen med för många tasks, något vi var tvungna att akta oss för flertalet gånger.

Efter insikten om att stycka upp user stories följde strax en annan: Att värdera tasks i timmar är problematiskt och otroligt svårt. Speciellt i vårt fall, med tanke på att vår startsträcka inför varje task varierade allt eftersom vi blev mer bekväma i utvecklingsmiljön. Vi förstod vid detta laget att det är bättre att vara konsekvent i sin värdering av tasks och justera velocityn efter vad man tror sig kunna uträtta under en viss sprint. De första sprintarna hade då fortfarande kunnat vara mindre

ambitiösa och värderingen hade inte behövt vara baserad på timmar utan siffror som representerar hur stor tasken var i termer av arbetsbörda. Värt att nämna är också att vi hade svårt för att acceptera att värderingen av tasken skulle vara en så pass abstrakt siffra. När vi väl förstod att denna abstrakta siffra (och värderingen av tasks överlag) endast är begriplig när man har gjort det några gånger och har en erfarenhet av vad den betyder, var det mer lätt att greppa.

Ytterligare ett feltramp som vi snart därefter skulle upptäcka var att vi inte delat upp våra user stories vertikalt. Denna felaktiga uppdelningen gjordes trots vi läst litteratur och gått på föreläsningarna som förklarar hur och varför de skall delas upp i vertikala tasks. Det visade sig helt enkelt vara mycket svårt att göra korrekt i praktiken. Vi tog oss därför en funderare över hur vi skulle ha kunnat dela upp tasks enligt tårtbitsprincipen och vertikalt. En anledning till varför att det var svårt att göra vertikala tasks var att de nu var så små att vi förlorade sammanhållningen av front- och backendaspekterna. Exempelvis var en task att endast skapa knappen, medan en annan var att koppla denna till någon funktion. Detta berodde i sig på våra svårigheter med att programmera i den helt nya miljön.

Samtidigt som vi åtgärdade våra felsteg lyckades vi även rätta till många andra problem än just uppdelning av user stories, som vi misslyckades med trots att vi var pålästa. Vi omestimerade velocity (genom korrekt värderingspricip istället för timmar) och utvecklade nya acceptance tests för varje ny task, samt en definition of done med checklista.

Vad vi tar med oss till nästa projekt är egentligen all erfarenhet som beskrivs ovan, det vill säga att user stories är **grunden** till hela scrum-metodiken. Det är med user stories varje scrumprojekt börjar och om det inte arbetas med user stories på ett seriöst sätt kommer det komma ikapp en under projektets gång. För att åstadkomma en stabil grund i nästa projekt bör vi rent konkret fokusera på att estimera och värdera arbetsbördan för user stories samt tasks, **inte** utefter hur många timmar de kan tänkas ta. Vi bör också fokusera på att dela upp user stories vertikalt, vilket hanterar både front- och backend, så att det är möjligt att visa upp något värdeadderande efter varje sprint. Dessa är de två största lärdomarna vi tar med oss. Förhoppningsvis tar oss detta arbetssätt dit vi vill vara: User stories och tasks är väldefinierade, estimerade i termer av arbetsbörda och uppdelade vertikalt.

Ytterligare bidragande till realiseringen av vår målbild till nästa projekt är att planera in mer tid till att bearbeta user storisen till en början. Här ingår även att kolla av med produktägaren, så att vi fått rätt specifikation av vad som ska göras. Vi måste även inse och öva på att dela upp user storisen vertikalt, samt estimera dessa. För att underlätta detta kan övningar som litteraturen beskriver tas till, exempelvis ”Splitting user stories - The hamburger method”.

4.2 Roller

The roles you have used within the team.

I samband med att gruppkontraktet gjordes fördelades roller ut. Följande roller delades ut: KPI-ansvariga, scrum masters, personer som skulle representera produktägare ("inofficiella produktägare"), reflektionsansvarig, rumbokare och sekreterare. I tabell 4.1 följer en beskrivning på de olika rollernas ansvarsområden.

Tabell 4.1: De roller som använts under arbetet samt beskrivning av dessa.

Roll	**Beskrivning**
Scrum masters	Hade som uppdrag att hålla i stand up-möten (daily scrums), att närvara på scrums of scrums, ha kontakt med andra grupper samt att se till att scrummetodiken efterföljs. Uppgiften delades på två personer för att uppnå rimlig arbetsbörda och ha någon att diskutera tankar med.
KPI-ansvariga	Ansvariga för insamling och uppföljning av de tre KPI:er som valdes i början av projektet. Här gavs en person ett KPI var för att underlätta arbetet och få en tydligare ansvarsfördelning.
”Produktägare”	Hade i uppdrag att hålla kontakt med den egentlige produktägaren i syfte att se till att produkten uppfyller de krav som produktägaren har, samt att kunna agera som produktägare på sprint reviewn.
Reflektionsansvarig	Var ansvarig för att se till att både gruppreflektionen och de individuella reflektionerna blir utförda varje vecka.
Rumbokare	Hade i uppdrag att se till att boka ett grupprum eller vara på plats i ett först-till-kvarn-rum till varje möte (ca 3 gånger/vecka).
Sekreterare	Ansvar att anteckna under varje daily scrum.

Rollerna hölls bra under den inledande delen av arbetet och skapade en tydlig ansvarsfördelning. När all tid gick åt till att försöka komma igång med utvecklingsmiljön och börja producera kod blev vi allt sämre på att följa scrum-metodiken och de agila verktygen. Gruppen tappade då även fokus på roller som produktägare, scrummaster och KPI-anvsarig som relaterar direkt till det agila arbetssättet. Vårt ostrukturerade arbetssätt som vi hade under denna period resulterade även i att de övriga rollerna föll bort.

När vi kom igång med programmeringen tog vi nya tag och återvände till det agila arbetssättet. Med det agila arbetssättet hittade vi också tillbaka till rollerna, till stor del genom att hålla ordentliga möten för retrospective. Något som vi upplevde

problematiskt var att rollerna blev aningen dynamiska, om en person missade att utföra sin uppgift täcktes detta upp av en annan. I efterhand anser vi att det hade varit bättre att påminna den ansvariga personen istället för att göra uppgiften åt den. Anledningen är att vi tror att rollerna långsamt suddats ut om vi hade fortsatt jobba på det sättet.

Det vi lärt oss utav detta och vill ta med oss i framtiden är att ha en tydlig struktur av rollerna i början och fortsätta jobba med dem kontinuerligt under arbetets gång. Detta bör göras genom att ständigt påminna varandra om de aktuella rollerna och inte täcka upp för varandra om rollerna missköts, vid behov kan rollerna dessutom behöva utvärderas gemensamt. Det är viktigt att alla har ett ansvar samtidigt som arbetsbördan är jämnt fördelad. För att utvärdera detta skulle exempelvis ett större utvecklingsmöte kunna hållas en bit in i projektet.

4.3 Agila metoder

The agile practices you have used for the current sprint.

Utformningen av projektets utförande har varit agil och vi har använt den agila processen scrum. De agila metoder som tillämpats är användningen av user stories, stand up meetings, korta projektiterationer (sprinter på en vecka), kontinuerlig testning, ett hållbart arbetstempo, parprogrammering och prioritering av kundvärde framför onödig dokumentation.

Till en början av projektet lärde vi oss om scrum och agil utveckling. Vi försökte anamma tänkandet och etablera de verktyg och rutiner som metoderna kräver. Verktygen som användes var en scrum board som från vecka ett fanns, i vilken vi la in några user stories som vi tänkte att Terminal 2 kunde tänkas behöva. Från första veckan av programmeringsprojektet etablerades också vanan att alltid ha stand-up meeting där samtliga i gruppen fick gå igenom vad de gjort, vad de tänkte göra och vad de eventuellt behövde hjälp med. Sprinternas längd var en vecka under hela projektet, och de korta iterationerna var en del av de agila metoderna som applicerades redan från start. Målet var att hålla en jämn arbetstakt och arbetstempo projektet igenom, och därför skapades en tidslogg och ett visuellt diagram, se 4.4. Vi sa tidigt att vi skulle utnyttja parprogrammering, vilket dock inte användes under de första veckorna då var och en istället försökte få igång utvecklingsmiljön på sin dator och alla hjälptes åt. Vi satte också projektmålet att leverera det som vår produktägare, Preem, värderade och behövde.

Ett par veckor in på projektet hade allas utvecklingsmiljöer installerats på de datorer som gick, och riktiga sprinter som behandlade programmeringstekniska user stories kunde påbörjas. Under den perioden fick vi dessutom in de user stories som Preem själva beskrivit. Dessa blev vår högsta prioritering och vi gjorde vårt bästa för att bryta ner dem och utforma dem så vertikalt som möjligt. Vi trodde att vi gjorde så som vi läst på och fått förklarat under föreläsningarna, men det visade sig senare att våra user stories var horisontella vilket påverkade det kundvärde som

första sprinterna levererade negativt. Under arbetet fortsatte vi med våra stand-up meetings, vilka fungerade bra och gav god överblick om vad alla höll på med.

Sprinternas start och slut var i början måndag till måndag. Detta fungerade inte bra ansåg gruppen då vi mötte produktägarna på onsdagar och det var mer rimligt att ha reviews och retrospective den dag med mest arbetstid till förfogande. Detta ändrade vi därför i mitten av projektet för att vi ansåg att det var ett naturligare avslut och början på onsdagar. Då träffades hela gruppen, hela dagen, och kunde tillsammans reflektera över föregående sprint och planera nästkommande, i samband med att vi kunde få handledning och prata med produktägarna. Tyvärr utvecklade vi inte i så högt tempo och kunde därför inte redovisa eller presentera varken speciellt stora eller kontinuerliga förbättringar.

Arbetsbördan försöktes hållas jämn, men antalet timmar som fanns tillgängliga varierade över veckorna och därmed likaså mängden arbete som lades ned under veckorna. Då allas utvecklingsmiljöer kom igång kunde parprogrammeringen appliceras på riktigt. De som hade en fungerande dator satt med någon som inte hade det. På grund av varierande närvaro var det dock inte optimalt med parprogrammering då det ibland bara var en eller ingen som arbetade på vissa tasks. Detta problem uppstod till följd av att alla inte alltid närvarade och att endast en i paret hade utvecklingsmiljön som krävdes, vilket gjorde det svårt att täcka upp för de som inte kunde närvara.

Mot slutet av projektet försökte vi göra om våra user stories så att de var mer vertikala efter att ha insett att de var horisontella, vilket var mycket svårare än vi trodde. Stand-up meetings fortsatte vi med kontinuerligt under hela arbetet, även mot slutet. Allt eftersom tiden gick blev vi duktigare och duktigare på att estimera hur mycket vi skulle hinna under sprintarna och kunde därför under de sista veckorna få ut maximal nytta av våra dialoger med produktägarna. Här pratade vi exempelvis med Henrik på Preem om hur vi skulle utforma våra user stories för att skapa så mycket värde som möjligt för dem. Mot slutet av projektet åtgärdade vi även problemet med parprogrammeringen genom att dela gruppen i två delar och låta en user stories behandlas av varje grupp. Detta på grund av att vi endast prioriterade två user stories samtidigt och då fanns det alltid minst en person som arbetade på båda. Till sist presenterade vi resultatet för Preem.

I framtida projekt är det av högsta vikt att komma igång med det praktiska utövandet så snart som möjligt. Detta eftersom att trots att vi läst på och trott oss förstå metodiken och programmeringsstrukturen, fick vi göra om en del saker då det ändå blev fel när det skulle göras i praktiken. Ett exempel är user stories som blev horisontella istället för vertikala. Parprogrammering var också lärorikt och något som kommer tas med till nästa projekt, eftersom det löste många av de problem vi hade genom att kunna bolla idéer mellan varandra. Parprogrammering är effektivt förutsatt att alla har utvecklingsmiljön och möjlighet att närvara. Vi upplevde även att det var enklare att lära sig det nya programmeringsspråket då man hade någon att bolla idéer med.

Genomgående under projektets gång har vi hållit oss från att göra ändringar under sprinten. Detta var bra på så sätt att det gjorde att vi till fullo försökte göra det som vi bestämt oss för under sprinten, men då vi hade dålig kunskap till en början om hur user stories skulle göras var det inte alltid rätt sak som gjordes. Det hade kanske därför varit bättre att i nästa projekt frångå metodiken och anpassat riktningen under sprinten, synnerligen med tanke på de begränsade antalet sprintar vi hade under detta projekt. Däremot lärde vi oss mycket genom att hålla ut sprinten igenom och istället tvingas göra bättre user stories och vara säkra på att vi ville genomföra just dem, samt att de var genomförbara, innan sprinten påbörjades. För att se till att detta sker även i nästa projekt bör vi lägga mer tid och vikt vid att diskutera genom user storiesen inplaneringen av varje sprint.

4.4 Spenderad tid på kursen

The time you have spent on the course (so keep track of your hours so you can describe the current situation).

I början av projektet belyste Håkan Burden vikten av att hålla ett kontinuerligt arbetstempo under kursens gång. Han nämnde då att 20 timmar i veckan (med justering för helgdagar) per person skulle hållas under hela kursen. Gruppens initiala mål var att lägga tiden som krävdes och då nå upp till sammanlagt 160 timmar varje vecka, men detta visade sig vara svårt bland annat på grund av att alla medlemmar samtidigt skrev sitt kandidatarbete. Påskledighet avbröt den andra sprinten som genomfördes och vi sänkte därför gruppens mål under den perioden.

När vi kom igång igen efter påsken ville vi ta igen lite av tiden vi förlorade i början av projektet och utökade därför gruppens tidsmål. Under den här perioden bytte vi dessutom sprintperiod från måndag-måndag till onsdag-onsdag, detta gjorde att en sprint blev extra lång. Mycket av tiden vi la ner under början och mitten av projektet gick åt till att få igång utvecklingsmiljön och förstå programmeringsspråket, det fanns därför en del frustration över att tiden vi la ner inte var värdeskapande.

I slutet av kursen gick samtliga gruppmedlemmar in i slutfasen av våra kandidatarbeten och behövde därför lägga ett större fokus på den kursen. Vi minskade då velocity för att matcha gruppens förväntade engagemang under tidsperioden. Slutligen la gruppen ner för mycket tid under den avslutande sprinten då appen skulle presenteras och vår slutgiltiga gruppreflektion skulle skickas in. Trots att både Håkan och Jan-Philipp tryckte på att vi skulle hålla ett hållbart tempo i slutet och inte sticka iväg för mycket så blev det en rejäl ökning av gruppens totala nedlagda tid. Detta faktum visar på hur svårt det faktiskt är att hålla en kontinuerlig, hållbar arbetstakt under ett helt projekt. Sammanfattningsvis har det också visat sig vara svårt att sätta ett gemensamt tidsmål när gruppmedlemmarna är beredda att lägga olika mycket tid på kursen. Det har även gjort det konsekvent svårt för oss att nå upp till det satta tidsmålet. Tiden som gruppen lagt ner illustreras i förhållande till

målsättningen i figur 4.1.

Figur 4.1: Bilden visar tiden som lagts ner varje sprint i förhållande till målsättningen.

För framtida projekt har vi lärt oss vikten av att komma överens om ett hållbart arbetstempo och följa detta. Ett problem under arbetets gång har varit att det under stunder har varit svårt att planera vad vi ska göra framöver. Svårigheterna i planeringen har lett till att ändringar i gruppens tidsmål har fått göras med kort varsel. Ett bättre sätt hade istället varit att gå igenom tidsramen för hela projektet innan, eller under, första sprinten. Eftersom vi visste att vi skulle ha mycket att göra med kandidatarbetet under perioder så skulle det, till exempel, varit typiskt bra att planera för det från början.

En annan faktor som troligtvis försvårat att hålla sig till en jämn arbetsbörda är det faktum att alla gruppmedlemmar samtidigt skrivit sitt kandidatarbete. Eftersom samma metodik med tidslogg och att eftersträva en jämn arbetsbelastning inte applicerats där, har det gjort att man behövt parera mellan detta projekt och kandidatarbetet. Om kandidatarbetet tar mer tid än vanligt är det troligt att en jämn arbetsbelastning över veckan prioriterats, vilket lett till att detta projekt blivit lidande istället. I framtida projekt har medlemmarna förhoppningsvis inte lika betydande åtaganden vid sidan av projektet, vilket gör att detta problem inte bör uppstå. Om så dock är fallet bör det i början av projektet klargöras vad som ska prioriteras av medlemmarna: en jämn arbetsbörda i detta projekt, eller en jämna arbetsbörda över medlemmarnas arbetsvecka. Därmed kan alla gruppmedlemmar ha samma förväntningar på övriga gruppens prestation.

4.5 Sprint Review

The sprint review (either in terms of outcome of the current week's exercise or meeting the product owner).

Under början av arbetet jobbade vi till viss del med sprint reviews. Vi gjorde den del av sprint reviewn som innebar kommunikation med produktägarna, men planeringen av sprinten var inte inkluderad. I detta skede hade vi ingen produkt att visa upp för produktägarna, men vi delgav istället våra idéer och fick respons på dem. Under kommande sprinter hade vi problem med att komma framåt i utvecklandet av de user stories vi hade visat upp. Eftersom vi inte gjorde några större kodmässigaframsteg att visa upp ansåg vi att det inte var någon idé att ha möten med produktägarna. Samtidigt som vi inte återkopplade till produktägarna planerade vi inte heller korrekt. Planeringen skedde inofficiellt i mindre grupper vilket gjorde att alla inte var involverade. Detta ledde till att vi jobbade ostrukturerat och hade otydliga mål inför sprinterna.

Anledningen till att våra sprint reviews blev lidande tror vi främst beror på att vi hade så stora problem med utvecklingen av koden. Detta skapade stress och vi kände att vi varken hade behov, eller tid att ha en sprint review, speciellt när vi inte kunde visa upp något. När vi under en längre tid hade stått still i vårt arbete kände vi som grupp att vi behövde ta tag i situationen. Vi införde de agila verktyg som vi lagt åt sidan och började arbeta aktivt med dem.

Införandet av sprint reviews gjorde att vi fick en bättre kontakt med produktägarna och började planera våra sprints bättre. Vi fick även möjlighet att utvärdera våra sprints och genom att vi delade med oss av våra erfarenheter undvek vi att göra samma misstag två gånger.

I framtida projekt är det av stor vikt att vi fortsätter med sprint reviews även när mer pressade situationer uppstår. Trots att det inte har skett några förändringar i koden bör vi ändå återkoppla med produktägaren för att få deras syn på situationen och möjligen hitta ett nytt scope för utvecklingen. Detta eftersom de kanske prioriterar annorlunda eller anser att det är bättre att bara jobba på en utav våra user stories om vi har problem med kodningen. För att göra detta bör vi planera in tid för reviews och eventuellt ha någon som ansvarar för att reviews sker.

4.6 Användningen av nya verktyg

Best practices for using new tools and technologies (IDEs, version control, scrum boards etc.).

För att genomföra projektet enligt scrum-metodiken och koordinera gruppen mot ett gemensamt mål behövde vi lära oss att använda nya verktyg. Ingen av oss har

någon tidigare erfarenhet av att arbeta med IT-projekt och vi behövde därför lära oss flertalet vertyg som för insatta utvecklare är allmänkunskap. Dessa verktyg kan delas upp i två grupper: *Verktyg för strukturering av arbete* och *Utvecklingsverktyg*.

4.6.1 Verktyg för strukturering av arbete

I denna grupp ingår verktyg som gjorde det lättare att arbeta på ett mer strukturerat sätt. Vi använde oss framförallt av Github, Trello och Slack. Hur vi gjorde det och vad de olika verktygen innebär beskrivs nedan.

4.6.1.1 Github

När vi på den första föreläsningen fick höra ordet ”repo” hade vi ingen aning om vad det var men fick relativt snabbt höra om Github, främst eftersom det var där som kursens information finns. Till en början gick det väldigt sakta framåt för oss och det enda vi egentligen la till på Github var våra veckovisa reflektioner. Det första stora problemet som vi ställdes inför var att flytta koden från portablecdm-repot till vårt eget repo. Detta gjordes av oss manuellt genom att Oscar la in hela appen i vårt repo.

Det generellt svåraste med att använda Github har varit att förstå hur brancher fungerar och hur merger skall ske. Under projektets mittendel lades tid på att läsa in sig på hur utvecklingen av brancher skulle ske och hur det skulle mergeas med varandra. Vi valde att ha var sin branch till en början där vi bara kunde testa att göra ändringar för att utforska och lära oss vad som gjorde vad. När vi senare började utföra tasks och user stories skapade vi istället egna brancher för dessa som sedan mergeades till masterbranchen. Innan mergen till master mergeade vi med en annan testbranch för att se hur mergen skulle fungera. Eftersom vi inte hade gjort större, radikala ändringar av appen gick alla våra merger relativt smärtfritt. Det var inte många tillfällen då vi behövde ångra en commit eller merge och vi behövde bara reverta en gång, vilket känns tacksamt i efterhand.

Efter ett tag insåg vi att vi måste ha en tydligare struktur på repot och skapade då mappar med tydliga namn för att bättre organisera oss och skapa ett effektivare arbetssätt. Allt eftersom projektet fortsatte kände vi oss mer bekväma med att använda versionshantering via Github och började förstå fördelarna med att använda verktyget. Lärdomar vi tar med oss till framtida projekt är dels användningen av verktyget i sig, men även vikten av att vara noga med versionshantering i projekt med många medlemmar. Tidigare har vi arbetat mer självständigt i mindre projekt och därmed inte haft samma behov av en tydlig versionshantering.

För att använda Github på det sätt vi vill i framtida projekt krävs att vi, i ett tidigt skede, skapar en bra struktur på vårt repo. Vi behöver även vara noga med hur vi använder våra brancher och se till att merga till masterbranchen oftare. Detta kan möjliggöra att vi kan upptäcka defekter snabbare och därmed enklare driva projektet framåt.

4.6.1.2 Trello

Trello har använts som en virtuell scrumboard. I Trello så har vi satt upp user stories, tasks, tillhörande acceptance criterias samt definition of done (DoD). När vi satte upp vår tavla i Trello gjorde vi detta med story mapping i åtanke. Vi ville behålla det helheltsperspektiv som story mapping skapar och vi skapade därför följande listor.

Tabell 4.2: Beskrivning av de listor som skapades i Trello. De listor som är markerade med ett '+' är listor som har adderats under ett senare skede.

Namn	**Beskrivning**
Notes	I denna lista samlades kort som innefattade information om hur vi jobbade i Trello. Till exempel hur våra user stories skulle vara uppbyggda och hur velocity skulle estimeras.
Product Backlog	I denna lista samlades alla user stories som inte var påbörjade.
User Stories In Progress	I denna lista samlades alla user stories som var påbörjade.
Sprint Backlog	I denna lista samlades tasks från de user stories som låg i User Stories in Progress. Dessa skulle avklaras till nästa sprint.
In progress	I denna lista flyttade vi de tasks som någon aktivt jobbade med. Detta gjorde att vi kunde undvika dubbelarbete.
+ Testing	I denna lista låg de tasks som väntade på att bli testade.
+ Verification	I denna lista låg de tasks som väntade på att bli accepterade av product owner.
Done	I denna lista låg de tasks som hade blivit testade och godkända av product owner.

Att ha listan "user stories in progress" var extra viktigt för gruppen. Genom att ha denna lista kunde vi behålla helhetsperspektivet för de tasks som låg i "Sprint Backlog". Något som kan ses i ovanstående tabell är att vi även valde att addera två listor. Listorna "Testing" och "Verification" adderades eftersom det uppstod behov av att kunna visa vilka user stories som stod inför dessa moment.

Ett problem som vi stötte på med Trello var att vi använde samma tavla till alla sprints. Eftersom Trello inte har någon versionshantering har vi därför inte kunnat kolla tillbaka och se vad vi har gjort tidigare sprinter. Utöver detta har arbetet med Trello fungerat bra men har i vissa delar av arbetet har vår tavla varit något ostrukturerad. Med detta i åtanke anser vi att vi i framtida projekt bör ha en

ansvarig för verktyget som strukturerar upp och skapar nya tavlor inför varje sprints. I övrigt vill vi fortsätta på samma sätt i kommande projekt och framför allt fortsätta jobba med story mapping.

4.6.1.3 Slack

Eftersom vi redan från början ville separera våra privata kommunikationsflöden med de vi skulle använda för projektet behövde vi ett verktyg för detta. Några individer i gruppen hade tidigare använt Slack till detta ändamål och det verkade därför lämpligt att använda det nu med. Från början sattes tre kanaler upp som var mer generella. Allt eftersom arbetet gick framåt uppmärksammades nya behov där en helt egen kanal lämpade sig väl. Vi ansåg att ämnen som vi frekvent pratade om och behövde kommunicera kring skulle få en egen Slack-kanal. Samtidigt som detta skapades ytterligare kanaler. Tabell 4.3 visar de kanaler som vi skapat i Slack.

Tabell 4.3: Kanalerna som skapades i Slack. De med ett '+' framför skapades efter behov av kommunikation kring ett specifikt ämne med tiden.

Kanal	**Beskrivning**
general	I denna kanal diskuterades generella och övergripande frågor kring projektet.
möten	Kanal för att diskutera när och var möten skall ske, ge eventuella återbud samt vägbeskrivningar till diverse salar.
random	Kanal för att diskutera saker som inte rör projektet.
+ estimate-velocity	Kanal som användes vid estimering av tasks. Alla skrev in vad de tyckte att en task skulle ha för estimat för att sedan debatteras ytterligare.
+ koddelning	Kanal för att kunna skicka kodstycken mellan varandra i syfte att öka kompetensen inom gruppen.
+ kpi	Kanal för att skicka ut formulär till välmående-KPI samt för att diskutera och följa upp alla tre KPI:er.
+ randommusik	Kanal som användes för att sprida musik i syfte att både öka den interna motivationen samt förbättra gruppens stämning.
+ redux	Kanal för att dela med sig av insikter och upptäckter som gällde designmönster.
+ teletype	Kanal för att dela de teletypekoder som vi använde för att skriva i Atom.

Med de nya kanalerna blev det lätt att hitta den information som var relevant, både ny och om det som skett innan. Efter tillägget av kanaler så såg vår Slack likadan ut till projektets slut och användes på det sätt som vi tänkt oss. Slack har varit ett

bra verktyg som även kan tänkas användas till framtida projekt. Vi vill ta med oss tankarna om mer specifika kanaler redan från början. Med den erfarenhet vi fått från detta projekt kommer det i framtiden bli lättare att strukturera kanaler. För att skapa en tydligare struktur hade vi redan från början kunnat utse en person som var ansvarig för utformning av Slack.

4.6.2 Utvecklingsverktyg

Denna grupp verktyg innefattar de som faktiskt användes för att direkt utveckla appen. Verktygen var Atom samt Android Studio, vilka presenteras har nedan.

4.6.2.1 Atom

För att göra den faktiska kodningen användes verktyget Atom som är en generell textredigerare. Vi hade vid projektets början inte någon koll på vilket program som lämpade sig bäst för att skriva JavaScript i eller vilket program som lämpade sig väl till Github-synkronisering. Atom var den första textredigerare som vi fick höra om som kunde versionshantera. Vi fick även höra att den skulle vara ganska simpel och fungera till projektet. Vi hittade fort funktionen teletype som möjliggjorde parprogrammering på så vis att det tillåter en annan person se och redigera en annans kod, likt Google Docs™.

Eftersom vi var oinsatta i koden och hur Github fungerade så användes Atom till en början enbart till att skriva våra reflektioner i. Vi märkte då att Atom var en bra textredigerare eftersom den förstod filformatet ”.md” vilket underlättade för oss. Att få Atom att samarbeta med Github var inte helt smärtfritt till en början då vi inte helt förstod hur commits och brancher fungerade. Vi läste på mycket om Github på diverse internetsidor samt YouTube och testade oss fram till en början. Efter att vi förstått detta var det inte längre några problem med att pusha till Github.

Ett problem kvarstod dock i Atom, men det var snarare ett problem i Github. Problemet låg i att vi inte blockerat vissa filer i vår .gitignore-fil. Detta gjorde att vi i början fick 3000 ändringar i filer då vi byggde om appen i Android Studio. Vi förstod till en början inte varför det blev på detta viset, men efter att ha konsulterat klassens Slack-grupp förstod vi att vi måste ändra i .gitgnore-filen. Efter att ha ändrat .gitignore-filen fungerade Atom bra.

Senare i projektet började vi som tidigare nämnt parprogrammera. För att underlätta detta integrerade vi insticket Teletype i Atom, vilket lät oss ”dela” kod mellan gruppmedlemmarna i realtid och därmed göra ändringar i varandras branches. Detta möjliggjorde dessutom att vi kunde programmera på datorer där vi inte fått Android Studio och själva simulatorn att fungera.

Till framtida projekt tar vi med oss att se till att endast de filer som ska pushas till Github kommer att göra det från början så att vi slipper problem med att rensa i koden och minimera risken för merge-conflicts. För att göra detta är det viktigt att vi redan från början är noga med att kolla igenom allt vi har i vårt repo. Vi

ska innan vi börjar programmera veta exakt vad alla filer på vårt repo innehåller och har till uppgift att göra. Vi tar också med oss möjligheten att göra ändringar i varandras branches genom att utnyttja Teletype som tillägg i Atom, eftersom det förenklade parprogrammeringen betydligt.

4.6.2.2 Android Studio

Android Studio användes för att ”bygga” appen. Till en början var det högst oklart om vi skulle behöva använda programmet över huvud taget men det klarnade när vi började förstå hur appen var uppbyggd. När vi väl förstått vilka verktyg vi behövde så laddades Android Studio ned. För att visa appen använde sig programmet av en emulator som visualiserade appen direkt på datorn. Emulatorn var väldigt tung och krävde en del datorkraft, vilket ledde till att den inte fungerade på allas datorer. Detta ledde i sin tur till att vi inte kunde programmera eller i alla fall köra emulatorn på alla datorer vilket minskade programmeringsmöjligheterna en aning. Å andra sidan så forcerades vi till att parprogrammera, något som vi visserligen ändå hade tänkt att göra.

Programmet hade speciella egenskaper som vi behövde lära oss, till exempel skapades temporära egna filer när appen ”byggdes”. När vi i början av projektet pushade dessa temporära filer, som är specifika för varje användare, fick vi väldigt många errors. Det tog några sprinter att lära oss att lösa dessa problem på ett smidigt sätt. Till slut lyckades vi även blockera filerna från att pushas till Github.

Emulatorn var bra men fungerade som sagt inte alltid på allas datorer. Mot slutet av projektet lärde vi oss fixa de enklaste felen som uppstod och vi allokerade gruppens resurser till de datorer som hade Android Studio som fungerade. Motivationen försvann ofta när programmet inte ville fungera och buggade. Vi lärde oss dock successivt att använda verktyget mer och hantera de problem som uppstod. Till framtida projekt så har vi nu byggt upp en grundförståelse för verktyget och kan förhoppningsvis komma igång snabbare med det nästa gång. En annan lärdom som vi tar med oss av projektet är att det krävs mycket tid och energi i början av ett projekt för att implementera alla utvecklingsverktyg. Därmed bör vi i nästkommande projekt se det som en del av uppstartsfasen att få igång utvecklingsmiljön på alla datorer, eftersom det sedan möjliggör ett effektivare utvecklingsarbete.

5
Övriga reflektioner

Arbetet med att utveckla Portable CDM har både varit en utmaning och utvecklande. Flera aspekter av arbetet har präglats av det faktum att projektet inneburit programmering som legat högt över vår kunskapsnivå samt att det tog flera sprinter för oss att få igång utvecklingsmiljön.

Detta har gjort att vi inte har kunnat utveckla så mycket som vi önskat och att vi under flertalet sprinter inte har gjort några framsteg i applikationen. De agila verktygen har på grund av detta inte kunnat utnyttjas på ett önskvärt sätt och har i delar av arbetet därför upplevts som en begränsning. Flera daily-meetings resulterade endast i att gruppmedlemmarna uttryckte frustration över att vi inte kom någonstans. Utan några framsteg i processen blev därför retrospectives och reviews överflödiga eftersom vi ändå inte hade något att se tillbaka på, reflektera över eller visa upp för produktägaren. Planeringen för nästa sprint bestod endast i att lösa de akuta förutsättningarna för att kunna börja programmera. Tillsammans ledde dessa faktorer till att scrum-metodiken avstannade under en period.

Att scrum-metodiken pausades på detta sätt ledde till ett mer ostrukturerat arbete, men samtidigt gav det oss den tid vi behövde för att sätta oss in i applikationen. I efterhand anser vi att det finns vissa delar ut scrum-metodiken som vi skulle behållit under denna period. Framför allt delar som ökade återkopplingen inom gruppen och som fick gruppen att jobba mot ett gemensamt mål. Under kommande arbeten hoppas vi att denna erfarenhet har hjälpt oss att undvika många av de fallgropar som vi nu stött på.

Appendix A - Acceptance tests

Generella tester:

- Testa funktionen själv/med din parprogrammerare och kontrollera att funktionen fungerar som ni tänkt.
- Låta någon annan i gruppen testar funktionen och denne ska ha förstått den nya funktionaliteten inom 5 min.
- Användaren (*Henrik/Sandra/Mathias*) testar funktionen och verifierar att user storyn är genomförd.

User story 1:

- ❑ **1.1 -** Specificera hur vyn ska se ut innan testningen påbörjas. Få vyn att visas genom appen, förslagsvis genom att slutföra 1.5 först. Se till att vyn ser ut enligt specifikation.
- ❑ **1.2 -** Hårdkoda in ett fartyg i vyn Mina Fartyg. Öppna appen och se till att fartyget finns i vyn. Om ja är testet godkänt.
- ❑ **1.3 -** Notera de fartyg som filtreras ut för kaj 521 med den ordinarie filterfunktionen. Gå in i inställningar, ställ in kaj 521. Se till att alla fartyg som tidigare noterades nu syns under Mina Fartyg. Upprepa testet för två andra kajplatser.
- ❑ **1.4 -** *Behöver mer förklaring och specifikation kring vad som menas.*
- ❑ **1.5 -** Se till att knappen för Mina Fartyg är synlig i menyn. Klicka på knappen och se till att en reaktion visas på skärmen, ex. att knappen trycks in
- ❑ **1.6 -** Se till att minst ett fartyg visas under Mina Fartyg. Öppna vyn för Mina Fartyg, kontrollera visuellt att ETA syns bredvid fartyget.
- ❑ **1.7 -** Öppna appen, se till att Mina Fartyg är den vy som initialt visas (*startsida*). Kontrollera att det från denna vy går att navigera sig till andra delar av appen, såsom inställningar. Utför **Generella tester**.

User story 2:

- ❑ **2.1 -** Koda in tre olika färger och kontrollera att fartygen i Port Call List har denna färgen.
- ❑ **2.2 -** Klicka på ett fartyg, se till att alla andra fartyg blir röda. Upprepa tre gånger genom att klicka på olika fartyg (*stickprov*).
- ❑ **2.3 -** Hårdkoda ett fartyg till att bli grönt. Kontrollera visuellt att detta fartyg blir grönt i listan av fartyg i appen. Utför en filtrering ur ordinarie filtreringsfunktion, kontrollera att samma fartyg fortfarande är grönt.
- ❑ **2.4 -** Koda ett nytt timestamp för ett fartyg. Kontrollera att samma fartyg som fick en ny timestamp har blivit rött i listan för fartyg.
- ❑ **2.5 -** Förklara för två andra gruppmedlemmar hur appen tar emot information om ETA. Om gruppmedlemmarna sedan kan återge denna process på ett korrekt sätt är testet godkänt.

- ❏ **2.6 -** Lägg till minst tre timestamps, antingen UNDER WAY eller PLANNED. Testa knappen såsom den är tänkt att användas. Kontrollera att alla röda symboler försvunnit.
- ❏ **2.7 -** Lägg in tre timestamps; UNDER WAY, PLANNED och BERTHED. Kontrollera att endast UNDER WAY och PLANNED får röd markering. Utför generellt test.

Appendix B - Definition of done

Checklists DoD

Definition of Done checklist for User Stories

1. User Story Clarity
    a. Everyone in the group understands the story and can rate its complexity (i.e hours)
2. Acceptance criterias identified
3. Tasks Identified
4. Project builds without errors
5. Unit tests written and passing
6. Project deployed on the test environment identical to production platform
7. Tests on devices/browsers listed in the project assumptions passed
8. Should look like Henrik's pic (if possible)
9. To-Do tasks completed
10. Feature is tested against acceptance criteria
11. Refactoring completed
12. Any configuration or build changes documented
13. Documentation updated
14. Peer Code Review performed
15. Regression Testing Done
16. Performance Testing Done
17. Feature ok-ed by Product Owner

Definition of Done checklist for Sprint

1. DoD of each single User story, included in the Sprint are met
2. "to do's" are completed
   All unit tests passing
3. Product backlog updated
4. Project deployed on the test environment identical to production platform
5. Tests on devices/browsers listed in documentation passed
6. Tests of backward compatibility passed
7. The performance tests passed
8. All bugs fixed
9. Sprint marked as ready for the production deployment by the Product Owner

Definition of Done checklist for Release

1. Code Complete
2. Environments are prepared for release
3. All unit & functional tests are green

4. All the acceptance criterias are met
5. QA is done & all issues resolved
6. All “To Do” annotations must have been resolved
7. OK from the team: UX designer, developer, software architect, project manager, product owner, QA, etc.
8. Check that no unintegrated work in progress has been left in any development or staging environment.
9. Check that TDD and continuous integration is verified and working