Reflektion kring arbetsprocessen

vid utvecklandet av applikationen SvenScan

Felix Jansson, Jesper Lindström, Lisa Larsson, Maija Happonen
Rebecca Finne, Samuel Håkansson, Simon Sundström.

Roller

Att arbeta effektivt i grupp är alltid en utmaning inom projektarbeten. För att underlätta med detta finns det vissa teorier och metoder att följa. I den här kursen användes scrum som arbetssätt.

När man arbetar med scrum finns specifika roller för att klargöra vem eller vilka som tar beslut och bär ett visst ansvar. De tre viktigaste rollerna är produktägaren, Scrum master och utvecklare.

Produktägare

Rollen som produktägare hade ej någon inom arbetsgruppen, utan istället master-studenter på "Interaction Design"-programmet (hädanefter benämnt ID), vilket gruppen anser har varit positivt för projektet. Produktägaren hade kunskap om design vilket underlättade kommunikationen med utvecklarna. De kunde förstå designen, komma med konkreta krav och förslag. Något som dock saknades var en mer aktiv, bestämmande roll då det gällde prioritering av uppgifter. ID hade kontakt med användare och besatt därmed kunskapen om vad användarna efterfrågade och borde därför haft en mer bestämmande roll.

Scrum master

Rollen som Scrum master blev aldrig direkt tilldelad någon individ. Detta var ett medvetet val som gjordes då Scrum master-jobbet innehåller arbetsuppgifter som inte behövdes i detta projekt då det är relativt kort. De arbetsuppgifter som fortfarande behövde utföras, så som att planera kommande sprint, sköttes istället genom gemensamma beslut i gruppen. Det enda som kan ha varit negativt med detta tillvägagångssätt är att det har kostat tid då det inte finns någon som har en tydligare översikt som man kan vända sig till då man behöver fråga saker om projektet.

Utvecklingslag

Roller inom utvecklingslaget, som till exempel testare och designers, ansåg vi inte vara nödvändiga på grund av projektets storlek. Gruppen ansåg också att det i lärandesyfte är bättre att varje gruppmedlem får möjlighet att utvecklas inom alla områden, istället för att ge enskilda personer ökad spetskompetens. Det har dock under tiden visat sig att olika personer i gruppen har haft en tendens att ta på sig olika roller på grund av olika förkunskaper och intressen. Tydligare roller hade dock förmodligen lett till bättre struktur och därmed mer effektivt arbete.

Kommunikation

För att underlätta kommunikationen och samarbetet så valde gruppen att avsätta tid varje vecka för att sitta tillsammans och arbeta. Att sitta tillsammans gjorde kommunikation och utbyte av information inom gruppen effektiv och lättillgänglig.

Ett problem som uppstod var dock att olika personers arbete var beroende av varandra och då en person arbetade utöver den utsatta tiden och behövde utökad eller annorlunda funktionalitet uppstod problem. Dessa problem ledde till att personen antingen måste avbryta sitt arbete halvfärdigt eller ändra i den andra personens arbete. För att lösa dessa problem så hade det underlättat att skriva tester först så alla vet vad de kan förvänta sig av alla komponenter.

Krismöte

Efter första sprinten insåg gruppen att det behövdes ett krismöte. Detta var på grund av att det hade hänt mycket i kodbasen utan någon direkt kommunikation mellan de olika arbetsgrupperna. Huvudsaken med mötet var att hitta en lösning till den bristande kommunikation och samarbete mellan de utvecklingsgrupper som jobbade på olika user stories. Slutsatsen var att dagliga scrum-möten är väldigt viktiga för att problem och annan information ska komma fram, och att alla ska känna sig delaktiga och att de hänger med på vad som händer i projektet.

Förändringen efter mötet var att regelbundna scrum möten återinfördes och att utvecklingslaget så ofta som möjligt skulle arbeta tillsammans på bestämd plats för att underlätta kommunikation mellan projektets olika delar och interna arbetsgrupper. Dessa förändringar bidrog till ökad effektivitet under kommande veckor och en positiv utvecklingskurva såväl för gruppen och utvecklingen av applikationen

Tillämpning av scrum-processen

Scrum board

Trello är ett virtuellt verktyg vilket användes i detta projekt för att skapa ett scrum board. Det bidrog till att projektets scrum board alltid fanns tillgängligt för alla gruppmedlemmar, även vid arbete hemifrån. Utvecklingsgruppen satte sig ned tillsammans inför projektets första sprint och konstruerade user stories. Det finns ett flertal praktiska funktioner i Trello såsom att tilldela kort till medlemmar i utvecklingsgruppen och skapa listor inuti dessa. Varje kort kan även kategoriseras med färger vilket underlättar vid navigation.

Det uppstod tidigt problem med projektets scrum board på grund av Trellos tekniska begränsningar för att visualisera detta samt att utvecklingsgruppen misslyckats med att bryta ned uppgifter. Till följd blev det för stora eller för komplicerade uppgifter vilket resulterade i att många user stories inte slutfördes under projektets första sprint.

Den funktionalitet gruppen saknade i Trello, som hade gjort det lättare att dela in user stories i mindre uppgifter, var att kunna tilldela uppgifter i checklistor under respektive user story till

specifika gruppmedlemmar. Till följd av detta skapade gruppen nya kort även för uppgifter, vilket skapade en visuell oreda och gjorde det svårare att urskilja vilka uppgifter som tillhörde vilken user story. Gruppen blev, som ett indirekt resultat av denna tekniska begränsning, mindre villiga att skapa nya uppgifter, då de kände att de mindre uppgifterna inte förtjänade ett helt eget kort i Trello. Dessvärre upptäcktes ingen konkret lösning på detta problem, och ett alternativ vid större projekt hade kanske varit att hitta ett mer komplett substitut till Trello.

Pappersprototyp och målbild

Gruppen tog inledningsvis fram en vision som skulle eftersträvas, samt diskuterade hur applikationen skulle se ut och vilken funktionalitet den skulle innehålla. Hela gruppen verkade överens gällande design, kodstruktur och flöden. Efter två iterationer skissades designen upp i samråd med ID. Det blev då tydligt att bilden av flöde och design inte stämde överens mellan utvecklingslagets medlemmar. Ett stort misstag som gjordes var alltså att gruppen inte tidigare i processen hade skissat upp en gemensam bild av applikationen, utan endast diskuterat muntligt. Detta medförde onödiga komplikationer till följd av att olika personer hade gjort olika antaganden, vilka borde ha klargjorts i ett tidigare skede.

Parprogrammering

Parprogrammering genomfördes ofta under projektets gång till följd av projektets mindre skala och dess få delar att arbeta med parallellt. Detta var i synnerhet praktiskt under projektets uppstart, då applikationens vyer och klasser var tvungna att skapas innan arbete kunde ske parallellt på dessa olika delar, och likaså att vi var tvungna att införa struktur i projektets kodbas. Många problem med kodduplicering uppstod innan behovet av parprogrammering insågs, dels på grund av dålig kommunikation och dels eftersom många var beroende av varandras funktionalitet och klasser.

I efterhand har det diskuterats hur problemet med parallellt arbete i småskaligt projekt hade kunnat lösas smidigare. Gruppen kom fram till att man kunde ha skapat gränssnitt och tester för funktionaliteten som behövdes i respektive sprint, för att sedan i självorganiserade smågrupper implementera valda delar. Detta hade medfört att man som utvecklare vetat

tydligare vad som förväntas av en själv respektive andra, och utöver det hade grupperna kunnat använda gränssnitten för metodanrop innan de konkreta klasserna skapades.

Test-driven utveckling och “continuous integration”

Test-driven utveckling har dessvärre inte genomförts i den utsträckning gruppen ursprungligen önskade. Detta beror främst på att applikationen till stor del bygger på system från tredjepart, så som textigenkänning och databastjänsten Firebase. Textigenkänning hade varit önskvärt att testa, men då den var så pass integrerad i Android och exempelvis krävde en specifik mappstruktur på enheten, lyckades vi dessvärre inte testa detta i ett fristående enhetstest. Gruppen försökte också införa gränssnittstester med hjälp av Android, men lyckades dessvärre inte få detta att fungera. Denna typ av testning hade eventuellt kunnat kringgå de problem som upplevdes gällande enhetstestning och problem att testa kod som beror på Android-plattformen, då gränssnittstesterna körs i en faktisk Android-miljö.

Ett försök att införa “continuous integration” (automatisk regelbunden körning av testerna) genomfördes också under projektets första sprint, men utan resultat. Gruppen lyckades inte med dess dåvarande kunskapsnivå och erfarenheter att få Androids tester att fungera med den valda integrationstjänsten “Travis CI”. Till följd av detta blev testning sekundärt i utvecklarlaget, då gruppen dels inte fick kontinuerlig respons på när utvecklingsversionen fungerade, och dels hade svårt att testa applikationen. Detta är något som gruppen med fördel hade kunnat göra annorlunda, då kontinuerlig testning är en viktig del av att, med större trygghet och säkerhet, släppa mjukvara löpande.

Sprintomdömen

Under projektets gång hölls tre sprintomdömen där ID gav feedback på applikationen. Deras åsikter påverkade utvecklingen på ett positivt sätt genom att påpeka defekter eller utvecklingsmöjligheter i såväl design som flöden. Deras feedback medförde att utvecklingslaget ändrade sina prioriteringar och user stories i lagets scrum board. Det är vanligt att det sker förändringar under en agil arbetsprocess vilket var bra att laget utsattes för. I slutändan hjälpte det gruppen även att fokusera på rätt funktionalitet och se vad som skulle prioriteras.

Sprintlängd

För detta projekt beslutade gruppen att en sprint endast skulle vara en vecka. I ett riktigt projekt hade det varit önskvärt med en längre sprintlängd. Utvecklingslaget önskade dock att genomföra ett flertal sprintar, trots projektets mindre storlek. Anledningen var att hinna med fler iterationer, vilket var bra för att lära sig den agila arbetsprocessen.

Nedbrytning av uppgifter och tidsuppskattning

I en agil arbetsprocess som scrum ligger fokus på att alltid kunna leverera användarvärde. Att inte klara av detta var ett problem som utvecklingsgruppen stötte på. Detta grundades i att gruppen inte hade lyckats korrekt uppskatta vilken effort olika user stories hade. Orsaken till problemet var gruppens bristande erfarenhet av att dela upp arbete till mindre uppgifter. I synnerhet var detta synligt under projektets första sprint då alla user stories som skapades blev väldigt stora. De saknade genomtänkta uppgifter och specifika krav som definierade exakt vad som skulle implementeras för att anses vara färdiga.

Gruppens lösning blev att flera user stories delades upp till mindre deluppgifter samt att krav specificerades i respektive user story. Förändringarna medförde att det blev lättare att förstå vad som ingick i varje user story, vilket därmed också gjorde det lättare att uppskatta hur mycket arbete som skulle krävas för att färdigställa den. Lärdomen av detta är att tid som läggs på planering innan implementationen påbörjas medför att gruppen sparar tid under projektets gång då de slipper göra ovannämnda förändringar senare.

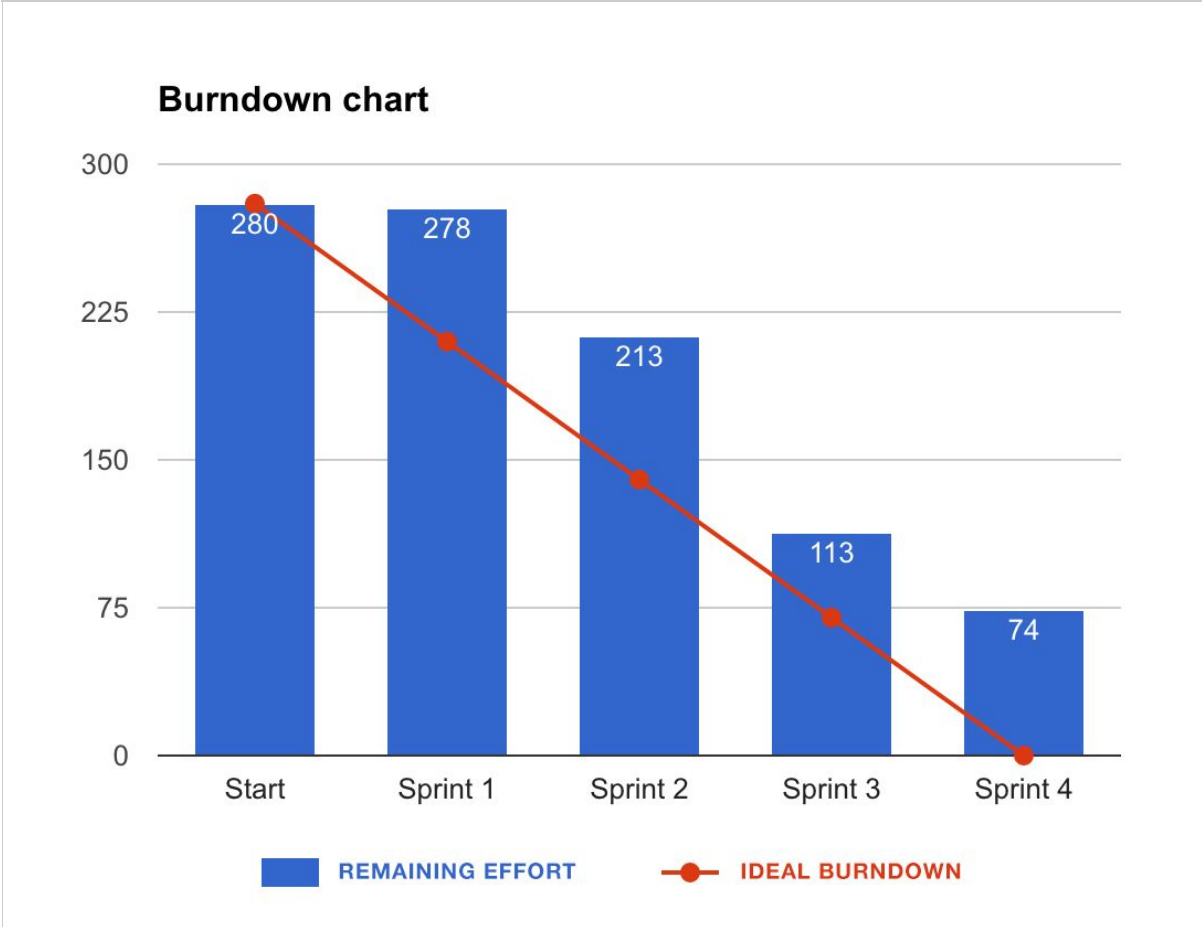
Att uppskatta en uppgifts effort var svårt, och dessvärre förbisåg utvecklingsgruppen helt de tekniska problem som kan uppstå. En lösning på detta hade varit att kontinuerligt utvärdera varje sprint och fastslå gruppens faktiska effort per uppgift. Genom detta hade gruppen kunnat göra mer realistiska uppskattningar.

Uppskattning av velocity

Gruppens velocity sattes till 70 inför första sprinten och baserades på att varje gruppmedlem förväntades leverera tio effort per sprint. Denna skala ändrades aldrig utan var konstant genom hela projektet. Fördelen med detta var att gruppen hade en statisk utgångspunkt att gå efter vid uppskattningen av effort för varje uppgift. Tanken att varje individ ska klara av att

leverera lika mycket effort varje sprint var dock felaktig. Den tanken medförde att det blev mer komplicerat att uppskatta effort på uppgifter, då olika individer i gruppen har olika förutsättningar för att lösa dem. En lösning på detta problem är att använda grupp, istället för individ, som minsta enhet och anse att gruppen i helhet ska leverera 70 effort. Att sammanställa och utvärdera gruppens velocity efter gångna sprint var dock något som uppdagades efter sista sprinten och fick då ingen inverkan på arbetsprocessen.

Burndown chart



Till en början levererade laget minimalt med effort, då uppgifterna ej bröts ner i tillräckligt små delar och tiden deluppgifterna tog underskattades. Detta var ett resultat av att alla var ovana att arbeta i utvecklingsmiljön och med den agila arbetsprocessen. Efter första veckan delades uppgifterna upp bättre vilket medförde en ökning av avklarad effort, vilket syns i grafen.

Under sista sprinten uppnåddes inte lika mycket effort. De uppgifter som kvarstod var antingen för stora och skulle därför ej bli klara i tid eller ansågs inte ge något värde till demonstrationen. Gruppen spenderade istället tiden med att göra buggfixar samt förbättring av gränssnitt och användarvänlighet, vilket tyvärr aldrig tilldelades någon effort. Detta val ledde till att utvecklingslagets burndown chart inte blev representativ av det arbete som gjorts under sista sprinten. En lösning till det problemet skulle vara att gruppen tilldelade effort på alla uppgifter som utfördes.

Tidsfördelning

Tidsåtgången för varje person i gruppen har varit ganska jämn. Den tid som alla enligt överenskommelse skulle lägga per vecka var sju till tio timmar. Sammanräknat spenderade alla gruppmedlemmar ungefär lika mycket tid men distributionen skiljde sig mellan veckorna på grund av olika personliga anledningar.

Att en gruppmedlem lade mindre alternativt mer tid en sprint var något som skedde. Ofta kompenserades detta naturligt av att individen spenderade mer eller mindre tid en annan vecka. Ett problem som uppstod med detta var dock att personen inte alltid fanns där för frågor om sin kod eller gemensamma beslut som skulle tas. På grund av projektets skala fick detta större konsekvenser än vad det hade fått i ett större projekt. Till följd av denna insikt valde gruppen dock att inte aktivt motarbeta detta problem.

Sprint retrospectives

Efter varje sprint har gruppen gjort en retrospective för att utvärdera sprinten. I dessa retrospectives nämns dels metoder kring arbetsprocessen som kunde gjorts bättre, samt vad som kan ha gått fel med implementeringen. Även den estimerade tiden för varje user story utvärderades, för att se om gruppen åstadkommit den effort som vår velocity krävde.

Sprint 1

Redan efter första sprinten märktes ett stort glapp mellan det som faktiskt hade implementerats och de user stories som uppfyllts. Detta berodde på att det inte ställts tydliga

krav på vad som skulle finnas med för att räkna en user story som färdigställd. Att gruppen började specificera dessa krav i varje enskild user story var något som ökade antalet klarade user stories varje vecka.

Sprint 2

Under denna sprint genomfördes det tidigare nämnda krismötet, vilket medförde förbättringar i kommunikation och hur vi använde vårt scrum board.

Sprint 3

Under tredje sprinten färdigställdes dels många kvarvarande uppgifter, samt nästan alla user stories som hade blivit utvalda, vilket därmed var en klar förbättring från föregående sprintar. Utöver detta upptäcktes andra problem, till exempel vad som ska göras då alla user stories var tilldelade och en individ blev klar med sin user story.

Sprint 4

Fjärde sprinten fick en mer genomgående utvärdering då tydligare utformade frågor ställdes, vilket medförde ett mer givande möte. Anledningen till att detta inte gjordes tidigare var att gruppen ansåg det ineffektivt att spendera tid på sprint retrospectives eftersom de oftast satt och jobbade tillsammans och ville återgå till utvecklingen av applikationen. Lärdomen blev att struktur på varje utvärdering skulle klargjort syftet med retrospectives, vilket skulle ha gjort dessa mer givande.

Anteckningar från retrospectives

Se all retrospective-dokumentation på GitHub.

Tekniska verktyg

Tredjeparts-bibliotek

Tredjeparts-bibliotek är en viktig del vid produktutveckling. En prototyp med komplex funktionalitet kan då skapas snabbt. Ett problem gruppen stötte på var att de tredjeparts-bibliotek som användes inte levererade efterfrågad funktionalitet till fullo. För att

motverka detta skulle gruppen från början behövt sätta upp vilka krav och funktioner som önskas samt läst på mer om biblioteket innan det implementerades. Dessa förändringar skulle ha medfört att gruppen funnit ett bibliotek som fungerade omgående istället för att spendera onödig tid på att testa ett flertal olika.

Versionshanteringsverktyg

Versionshanteringsverktyget git har använts genom hela projektet. Arbetsflödet med git har följt feature-branches vilket innebär att applikationen är uppdelad i olika versioner, där en ny version skapas för varje ny större funktionalitet som ska implementeras, för att slutligen kombineras till en komplett version efter varje sprint.

Det finns en så kallad master-branch vilken alltid bör innehålla stabil kod med färdiga och testade funktioner. Utifrån denna finns sedan en develop-branch (hädanefter benämnd "develop") där utvecklingen av applikationen utgår ifrån. Dock är det viktigt att inte alla nya funktioner implementeras direkt i develop, utan att varje ny funktion skapas i sin egen feature-branch för att sedan sammanfogas med develop när den är fullt implementerad och testad. Detta medför att det blir enkelt att arbeta parallellt utan att varandras halvfärdiga funktionalitet påverkar övriga arbetsgrupper.

Gruppen eftersträvade att för varje user story skapa en ny feature-branch. Det visade sig dock tidigt i projektet att missförstånd uppstått kring när och hur ny funktionalitet skulle publiceras i develop. Problematiken som uppstod till följd av detta var att halvklar funktionalitet publicerades i develop tidigare än önskvärt, och således orsakade komplikationer vid vidareutvecklingen för övriga arbetsgrupper.

Till en början användes feature-branches regelbundet, men mot slutet av projektet förblev utvecklingen i develop. Detta var då de flesta user stories var färdiga och det som var kvar att göra var mindre uppgifter eller buggfixar. Att då skapa en ny branch för detta kändes onödigt. Till följd av detta kunde problem uppstå så som att develop ej blir kompilerbar då icke-testad kod publiceras. Dock upplevdes aldrig större problem, men lärdomen är trots allt att man borde ha följt god praxis genom att inte utveckla i develop, då detta hade medfört större säkerhet under mjukvarans utveckling.

Relationen mellan processen och prototypen

Synpunkter från ID applicerades kontinuerligt under projektets gång. Detta fungerade väl tack vare den agila arbetsprocessen. Genom att dela upp projektet i tårtbitar och leverera en i taget skapas tidigt något att visa upp som därefter kan byggas vidare på. Om processen istället hade varit av “vattenfalls”-karaktär så hade det ej funnits något konkret för en produktägare att ge feedback på. Det hade då dessvärre endast funnits en sockerkaksbotten och ingen hel tårtbit. Nackdelen är då att all feedback som kommer från produktägaren inte kan påverka utvecklingen innan slutet av utvecklingstiden då mycket refaktorering antagligen skulle behöva göras.

Lärdomar från gästföreläsningar

Spotify

Gästföreläsningen med Spotify handlade om deras hierarki och hur de arbetade i lag som i sin tur bestod av grupper. Det är gruppen som bestämmer hur produkten ska utformas, inte chefen. Vår produktutveckling har skett likt Spotifys. Utvecklingsgruppen har inte haft någon specifik person som haft övergripande ansvar, utan alla har själva tolkat uppgifterna och kommit med förslag och kreativa idéer. Detta är inte något som vi har implementerat efter föreläsningen, utan det har snarare skett naturligt inom gruppen då ingen tilldelats någon specifik roll. Det var dock nyttigt att få veta att de jobbar på ett liknande sätt och att man använder liknande metoder i näringslivet, då detta visar på att vi varit på rätt väg med vår tillämpning av en agil arbetsprocess.

Volvo

Volvo pratade om deras process där de jobbade semi-vattenfall och semi-agilt samtidigt, på grund av att de i fordonsindustrin inte kan jobba agilt med mekanik på samma sätt som med mjukvara. Gruppen ansåg inte att detta var något som var relevant att applicera i detta mjukvaruprojekt, men förstod trots allt att Volvo och andra större företag upplever andra utmaningar och begränsningar med scrum, jämfört med mindre mjukvaruprojekt.

AR & HCI

Gruppen hade stora förväntningar på föreläsningen om AR & HCI då det planerades att AR skulle användas i applikationen. Föreläsningen innehöll dock endast material som gruppen redan tagit del av via tidigare kurs om grafiska gränssnitt, samt för projektet irrelevant information om AR. Gruppen förväntade sig mer genomgång om tekniker och konventioner inom AR som hade varit användbara för utvecklingen av applikationen.

The Techno Creatives

The Techno Creatives pratade om vad som kretsar kring dem som teknikföretag. Det pratades bland annat om hur företaget ser på pengar, rekrytering och framtiden. Däremot diskuterades inte vilka processer de använder sig av för att utveckla mjukvara, vilket gruppen hade velat få information om för att kunna förbättra gruppens process.

Utvärdering av D1A och D2

D1A

I D1A reflekterade gruppen kring Lego-övningen som genomfördes för att testa en agil arbetsprocess. Lärdomarna från övningen var vikten av god kommunikation med produktägaren tidigt i projektet, samt kommunikation och kontinuerlig utvärdering inom utvecklingsgruppen. I projektet applicerades lärdomarna genom att regelbundna möten med ID hölls, samt att utvecklingsgruppen eftersträvade att hålla dagliga scrum-möten och regelbundna sprint retrospectives.

Mötena med ID skedde ungefär en gång i veckan, och under första mötet presenterade ID flera produktidéer som gruppen kunde välja mellan. Inom varje produktidé gavs inga strikta direktiv utan bara en önskan om att få komma med synpunkter genom processen. Detta ledde till att lärdomen om underliggande krav blev irrelevant för projektet. Gruppen har i efterhand kommit fram till att fler möten med ID, samt att de antagit en mer konkret roll som produktägare, hade varit att föredra ur ett lärandeperspektiv och för projektets utveckling.

Under projektet applicerades korta regelbundna scrum-möten för att få överblick av vad resterande gruppmedlemmar arbetar med eller om någon behöver hjälp. Då gruppen inte förstod nyttan av dessa i början, så minskade frekvensen av mötena. Detta medförde kommunikationsproblem vilket kort därefter ledde till att mötena återinfördes, i enlighet med tidigare redogörelse om gruppens krismöte.

För att kontinuerligt utvärdera arbetet användes sprint retrospectives. Utvärderingen blev bättre och fick mer struktur för varje sprint som gick, men kunde ändå gjorts bättre. Framförallt borde mötena haft en tydligare mall och en specifikation på vad som gruppen skulle få ut av utvärderingen.

D2

I D2 diskuterades problem som uppstått som resultat av de slopade dagliga mötena, vilket resulterade i att regelbundna scrum-möten återupptogs igen mot slutet av projektet. Detta gav önskad effekt och ledde till en bättre förståelse för vad som skedde i projektet och därmed ett bättre samarbete.

Under half time review visualiserades applikationen och dess flöden vilket gav gruppen en gemensam bild att arbeta mot. Efter detta minskade osäkerheten kring gränssnitt, interaktionsflöde och kodstruktur. Mindre osäkerhet ledde även till en bättre stämning inom gruppen.

I D2 drogs även slutsatsen att en vecka per sprint egentligen är en för kort tidsperiod, då det kändes stressigt. Sprintlängden ändrades dock aldrig på grund av projektets storlek, enligt tidigare nämnda anledningar.

Efter D2 ändrades flera user stories genom att de delades upp i mindre delar. Dessutom gjordes nya, mer genomtänka uppgifter med krav för att förtydliga vad som krävdes för en user story. Dessa krav var sedan tänkta att användas för automatiserade tester, vilket visade sig vara svårare än vad gruppen förväntade sig. För att kunna leverera mervärde valde gruppen att ej genomföra automatiska tester utan förlita sig på användartester istället. Trots detta var det ändå till hjälp att ha tydliga krav för att utvecklaren enklare ska veta när en user story ska anses klar.