Post Mortem Rapport

FitnessTracker

Software Engineering – DAT255

2014-06-05

Författare, Grupp 5: Matilda Berntsson, Jenny Larsson, Joakim Berg, Michael Nordström.

Projektet

Projektet som denna rapport berör innefattar utvecklingen av en Android-applikation som döptes till FitnessTracker. Applikationen ämnar ge användaren ett verktyg för att hålla koll på sina utförda träningspass samt sitt matintag i syfte att ge användaren en möjlighet till att kontrollera sin prestation gällande fitness. Utvecklingen av applikationen utfördes under cirka åtta veckor, mellan den 24/3 till den 5/6 vårterminen 2014. Projektet utfördes vid Chalmers tekniska högskola i kursen Software Engineering DAT255.

Utvecklarna

Utvecklingsgruppen som utförde projektet består av fyra studenter från Chalmers tekniska högskola under handledning från Morgan Ericsson. Gruppsammansättningen utgörs av två elever från årskurs 3 på Civilingenjörsprogrammet Datateknik samt två elever från årskurs 3 på Civilingenjörsprogrammet Industriell Ekonomi.

Förhoppningen med denna gruppsammansättning var att skapa en projektgrupp med tvärvetenskaplig kompetens där dessa olika kompetenser kunde kombineras för att få ett så lukrativt projekt som möjligt. Syftet var bland annat att medlemmarnas differerande kunskaper och tidigare erfarenheter skulle kunna komplettera varandra på ett sätt som skulle generera ett bättre utfall i projektet samt att förbättra lärandet genom ett utbyte av kunskap gruppmedlemmarna sinsemellan. Exempelvis har medlemmarna från det Datatekniska-programmet mer erfarenhet gällande programkodutveckling samt viss kunskap i Android-utveckling, medan medlemmarna från Industriell Ekonomi har något mindre programkodsutvecklingsvana (restrikerat till Java) och saknar erfarenhet av Android-utveckling, men har mer erfarenhet av att arbeta i projektgrupper samt av exempelvis rapportskrivande och liknande. Därför var förhoppningen att dessa olika erfarenheter skulle komplettera varandra i ett projekt där samtliga bitar var nödvändiga för ett lyckat projekt.

Applikationen

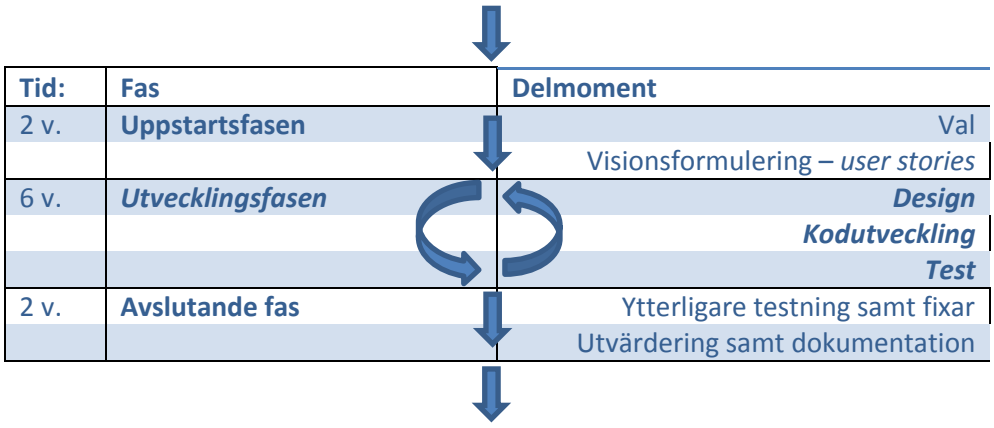
Applikationen döptes, som tidigare nämnts, till FitnessTracker och den huvudsakliga tanken bakom applikationen var att ge användaren ett bra samt lättanvänt verktyg för att hålla koll på sina prestationer gällande fitness. Applikationen skulle ge användaren ett verktyg för att kunna se vilken kalorimängd som har intagits i matväg samt vilken kaloriförbrukning som har medförts av utförda träningspass. På så sätt skulle användaren kunna se det totala antalet kalorier som intas och förbrukas per dag och därför även kunna se prestation över tid. Tanken var även att användaren skulle kunna definiera någon form av viktmål och att applikationen skulle kunna ge användaren information kring vilken kalorimängd som är lämplig att inta och förbruka per dag för att uppnå detta mål samt vilken tid det skulle ta. På så sätt skulle användaren kunna se sin dagliga prestation mot det uppsatta målet för att få bättre kontroll över hur denne ligger till samt över vad som krävs för att uppnå målet.

Andra viktiga egenskaper som applikationen ansågs behöva ha var att kunna lagra inmatad information på dagsbasis, så att användare skulle kunna se tidigare dagars resultat, och ge någon form av visualisering av prestation vid vald dag. Det var även extremt viktigt att applikationen skulle ha tillgång till data kring olika maträtter och träningsformer, så som kalorimängden dessa responderar till, då detta var grunden till det flesta funktioner som applikationen förväntades utföra. Syftet var att användaren skulle, vid inmatning, få förslag på en maträtt eller träningsform för att

förenkla användningen. Om en maträtt eller träningsform inte existerade skulle det finnas möjlighet för användaren att lägga till dessa som alternativ.

Mer specifik information kring applikationen, dess uppbyggnad, funktioner och design återfinns i projektdokumentation och visionsdokument.

Processen – en översikt



Projektets process bestod av ett antal delmoment och uppgifter som utfördes under arbetets gång. Dessa skulle kunna kategoriseras som uppstartsfasen, den huvudsakliga utvecklingsfasen samt den avslutande fasen.

Uppstartsfasen i detta fall bestod bland annat av *valet* av vilken applikation som skulle utvecklas. Valet av applikation skedde med hjälp av två stycken brainstorming-sessioner där gruppens medlemmar träffades och tog fram förslag. Vid den första sessionen var syftet att ta fram idéer av en bredare karaktär kring vilken typ av applikation som gruppen ville utveckla, i syfte att trigga tankeprocessen. Efter detta bestämdes att varje gruppmedlem skulle ta fram minst ett förslag till nästföljande session. Av dessa skulle tre stycken förslag sedan sållas ut vilka i sin tur skulle vidarebefordras till handledaren för ytterligare konsultation kring vilket förslag som var lämpligast. De förslag som sållades ut var en fitness-applikation, en kart-applikation som visade närliggande events och restauranger samt en matematik-applikation som genom fotografering av en formel kunde känna igen samt beräkna matematiska formler. Efter konsultation med handledare röstade gruppen om vilket förslag som ansågs mest lämpat för den givna tidsramen, med avseende på bland annat svårighetsgrad samt kurskrav, vilket resulterade i valet att utveckla fitness-applikationen FitnessTracker.

Nästa delsteg i uppstartsfasen var att formulera en *vision* gällande applikationen där bland annat möjliga användare och deras tänkbara krav och önskemål på applikationen skulle definieras. Denna togs fram genom att först definiera vilka användare applikationen skulle riktas till samt vilka egenskaper dessa användare hade för att i sin tur kunna formulera vilka krav och önskemål dessa användare kunde tänkas ha. Detta ledde fram till visionen samt till en nedbrytning av visionen i så kallade *user stories*.

User stories är formuleringar utifrån användarens perspektiv som beskriver vad användare vill kunna göra med applikationen. Syftet med dessa är att skapa en grund för ett kund-orienterat utvecklingsförfarande som innebär att applikationen utvecklas med avsikt på användarens önskemål

och krav. Det möjliggör ett *agilt* arbetsförfarande där funktionalitet och designval kan ändras efterhand. Huvuduppgiften bryts ner i deluppgifter som utförs under kortare etapper för att i sin tur uppfylla visionen med slutprodukten. Dessa agila utvecklingsmetoder är ofta *iterativa och inkrementella* för att främja en flexibel process med möjlighet för specifikationsförändringar. Med detta agila arbetssätt fanns således möjligheter till att lägga till ytterligare user stories senare under projektet, även om de allra flesta, som specificerats i visionen, formulerades redan i uppstarten.

Uppstartsfasen övergick sedan i utvecklingsfasen. Då projektgruppen eftersträvade en agil process bestod utvecklingsfasen utav iterativa delcykler som upprepades tills projektet var slutfört. En delcykel bestod av tre stycken delmoment, karaktäristiska för agila processer, nämligen *design*, *kodning* och *testning* (eng: *design-code-testing cycles*). Metodiken som har använts i utvecklingsprocessen beskrivs närmare under rubriken *Utvecklingstekniker*.

Det första beslutet som fattades gällande applikationens utformande var att en databas skulle behöva användas. Applikationens grundläggande design skelett bestämdes även tidigt, i form av en slidingmeny, där fragment med funktioner kunde utvecklas och läggas till. Efter det användes user stories för att driva fram utvecklingen ett moment i taget. Medlemmarna tilldelades en eller ett antal user stories som skulle designas och utvecklas vilket skulle avslutas med testning av den framtagna enheten. Efter en cykel hade avslutats skulle en ytterligare cykel påbörjas (mer detaljer kring delstegen som utfördes i utvecklingsfasen återfinns i projektets git repository samt i övrig projektdokumentation).

När utvecklingsfasen för detta projekt fullbordades påbörjades den avslutande fasen som främst bestod av projektutvärdering, presentation samt färdigställande av dokumentation. Utvecklingsfasen fortsatte delvis fram tills projektdeadline för att finslipa produkten samt fixa eventuella buggar.

Tidsfördelning

Projektet utfördes på halvfart, det vill säga med avsikt att varje gruppmedlem skulle lägga ner minst 25 timmar i veckan. Detta innebär en total arbetsmängd på 200 timmar per medlem och därav totalt 800 timmar.

De olika user stories som har slutförts har värderats med avseende på deras vikt för detta projekt i dokumentet *Product Backlog*. Där har även dessa user stories värderats med avseende på hur krävande de varit att slutföra, vilket kan ge läsaren viss information kring vart tyngdpunkten har legat i projektet.

Uppstartsfasen tog cirka två veckor och bestod till största del av utformningen av applikationens vision samt user stories men viss tid lades även på att få igång ett git repository samt på att få Android att fungera på samtliga medlemmars datorer.

De aktiviteter som var mest tidskrävande var framförallt utvecklingen av databashanteraren. Framtagandet av denna var dock av högsta prioritet för att möjliggöra utvecklingen av övriga funktioner. Det var därför av största vikt att denna utfördes så tidigt som möjligt och att den utformades på ett så smidigt sätt som möjligt, då detta skulle förenkla resten av arbetet. Detta utfördes av Matilda Berntsson som fullbordade uppgiften den 13e april, ett arbete som tog cirka 90 timmar att slutföra.

Övriga huvudsakliga aktiviteter var att skapa en sliding menu (ca 30h), finna data kring träning (ca 2h), BMR-beräkningar(ca 2h) och livsmedel (ca 2h), läsa in dessa i databasen (ca 25h), programmera fragment för matintag (ca45h), för utförda träningar (ca45h), för uppsättande av mål (ca 45h), samt en startaktivitet för inmatning av grundläggande användarinformation (ca 30h). Det behövdes även fragment som lade till träning respektive mat i databaserna (ca 30h st.), ett fragment som visualiserade dagens prestation och inmatningar (ca 30h), ett fragment som visade ett schema där ett klick på en dag visade home-fragment för den dagens inmatningar med mera (ca 28 h). Övriga aktiviteter var exempelvis layout och grafik samt integrering, buggfixning, dokumentation, testning samt viss övrig funktionalitet (se https://github.com/matbern/FitnessTracker för mer detaljer kring utförandet).

Den avslutande fasen tog cirka två veckor och bestod framförallt av dokumentering, utformande av denna Post Mortem-rapport samt planering av presentation.

Matilda Berntsson: Skapade hela databashanteraren. Skapade samtliga entiteter i databasen (dess tabeller). Skapade fragmenten för mat och träning (där sökning på matsorter/träningsformer gjordes i databasen samt där inmatning och lagring i databasen skedde). Utförde inläsningen av dataark till databasen och skapade ett mål fragment samt ett fragment för att ändra eller ta bort mål. Skapade ett fragment för att användaren skall kunna lägga till nya maträtter och träningsformer i databasen och skapade även ett fragment som visade vilka träningar och maträtter som blivit tillagda i databasen på ett specifikt datum (väljs med hjälp av en datepicker). Skapade JUnit tester för databasen. Formulerade vision, user stories, PM, samt Product Backlog. Hittade data för BMR och viktminskningslogik. Skrev beräknande kod för BMR samt hämtade data från mål, mat och träningsfragment för logiken i homefragment. Fixade viss layout.

Michael Nordström: Skapade ett kalenderfragment och en sliding-meny. Skapade ett homefragment och dess grafik. Hade huvudsakligt ansvar för grafiken och XML-kod i applikationen och dess layout. Skapade första aktiviteten för inmatning av användarinformation. Fixade vissa buggar och hittade livsmedelsdataark. Bidrog med viss Black-Box testning.

Joakim Berg: Skapade ett data fragment som listar träningsformer och ger information gällande dessa. Fixade vissa buggar, grafik och layout. Skrev Developer Documentation och Definition of Done och Black-Box tester.

Jenny Larsson: Skrev User Manual, hittade träningsdataark. Fixade buggar. Ändrade språk till svenska från engelska.

Utvecklingstekniker

I projektet användes framförallt principer ifrån utvecklingsmetoden Scrum, men även vissa delar av metoden eXtreme Programming, vilka beskrivs i detta kapitel.

EXtreme Programming:

EXtreme Programming är en agil utvecklingsmetod som baseras på fem värderingar nämligen: *kommunikation*, *enkelhet*, *återkoppling*, *mod* och *respekt*. Kommunikation innebär mellan utvecklarna i teamet men framförallt mellan utvecklarna och kunden. Det är därför viktigt i denna metodik att kunden är på plats så denne kan bestämma vilka krav som systemet skall uppfylla och även prioriteringen av dessa. Enkelhet fås genom kontinuerlig omstrukturering av kod samt genom

minimal användning av dokumentation och liknande. Återkoppling fås genom kontinuerliga enhetstester samt genom flera delleveranser. Mod innebär i detta sammanhang att våga göra det som är rätt trots att det inte är ett populärt beslut. Respekt handlar i sin tur om att respektera medutvecklarna genom att exempelvis inte ändra i någon annans kod utan god anledning som understötts med testning. Dessa fem värderingar uppfylls i sin tur med ett antal tillämpningar, varav några är planeringsspelet, enkel kod, parprogrammering, små leveranser och gemensamt ägarskap.

EXtreme Programming var en metod som initialt var ett alternativ för utvecklingen av applikationen. Den valdes dock i huvudsak bort på grund av svårigheter att finna möjlighet till och tid med att programmera i par. Detta då gruppmedlemmarna studerade olika program och därför även hade olika scheman vilket gjorde det svårt att finna gemensamma tillfällen för parprogrammering. Därför beslutades det relativt tidigt att denna metod inte skulle användas i någon större uträckning, men gruppen bestämde ändå att vissa tillämpningar kunde användas vid vissa specifika tillfällen i projektet.

De tillämpningar som användes från eXtreme Programming-metoden var exempelvis parprogrammering, även om detta endast var i mindre mån. Parprogrammering används i eXtreme Programming med avsikt att den ena programmeraren skall skriva koden medan den andra kontinuerligt granskar koden för att säkerställa enkelhet och god kodkvalitet. Fördelar med detta sätt att arbeta är att det är lättare att upptäcka fel eller brister direkt vid kodningen och därför är det ett bra sätt för att säkerställa god kodkvalitet. Det är även ett bra sätt för samtliga gruppmedlemmar att vara väl insatta i koden och tanken bakom, då fler medlemmar är med och utvecklar den tillsammans, vilket kan vara en fördel då medlemmarna kanske har mindre erfarenhet kring det som skall utvecklas eller är osäkra på hur lösningen bör utformas.

Nackdelar med parprogrammering är exempelvis att det är betydligt mer tidskrävande, då endast en utav medlemmarna kan skriva kod, vilket i teorin halverar mängden kod som kan skrivas per tidsenhet. Det kan även finnas viss problematik kring möjligheterna att koordinera arbetet så att medlemmarna fysiskt kan träffas och arbeta, särskilt om dessa har olika scheman. Detta arbetssätt är därför inte särskilt effektivt i de flesta fall, speciellt inte då programmeraren är relativt säker på koden och då risken för kvalitetsbrister är mindre. Dessa anledningar var två utav huvudorsakerna till att parprogrammering inte användes i någon större utsträckning i projektet. De tillfällen som arbetssättet användes var främst då medlemmarna stötte på någon form av problematik kring kodningen, eller då det fanns större behov av testning. Det användes framförallt i slutet av utvecklingen då mycket av grafiken skulle utvecklas. Detta var givande eftersom applikationen vid denna tidpunkt var något större och mer komplex vilket gjorde att det var lättare att missa någon detalj där grafiken skulle ändras. Därför var det användbart med parprogrammering för att säkerställa att allting blev rätt. I dessa fall var parprogrammering mer effektivt då det minskade tiden som annars skulle ha krävts för felsökning av samma person som skrev koden. Det användes även för en del buggfixar och testning i slutet av projektet, utav i princip samma anledning som gällande grafiken.

I ett framtida liknande projekt skulle detta arbetssätt användas i de fall där vikten av kodkvalitet hade varit stor eller där risken för fel hade varit stor, till exempel vid mer komplexa uppgifter. När mycket tid behöver läggas på att få enkel och felsäker kod är detta arbetssätt mer effektivt än att programmerarna kodar själva eftersom felsökningen sker parallellt med utvecklingen. Det är även

användbart i fall då utvecklarna är mer osäkra på hur de skall gå tillväga med en uppgift då de i dessa fall kan bolla idéer med varandra samt då en utav utvecklarna skulle kunna söka efter lösningar på problem medan den andra felsöker eller kodar.

Övriga tillämpningar från eXtreme Programming som har använts i projektet är gemensamt ägarskap, det vill säga att samtliga medlemmar har rätt att ändra i koden, vilket har varit en självklarhet i detta projekt. Med denna princip har det dock varit viktigt att man gör detta med respekt och fördelaktigt med god kommunikation kring varför ändringar har gjorts. Detta har fungerat bra i projektet och ingen problematik kring detta har uppkommit. Det har möjliggjort effektiva fixningar av koden och i ett framtida projekt hade denna princip använts så länge den utövas med respekt och kommunikation.

Andra tillämpningar från eXtreme Programming såsom *Test Driven development*, *programmer welfare* samt *Gemensam kodstandard* användes inte i detta projekt. Test Driven development innebär att tester skrivs innan koden utvecklas, av utvecklaren för att testa och validera koden (white-box-testing) samt av kunden för att validera vad koden skall utföra (black-box-testing). Detta hade naturligtvis varit önskvärt, både för att förbättra slutprodukten och i lärande syfte, men tyvärr ansåg gruppen att detta hade blivit något för komplext samt för tidskrävande att klara av. De tester som gruppen använde beskrivs senare i rapporten. Programmer Welfare innebär att arbetet skall utföras under 40-timmars veckor för att minska risken för utbrändhet och liknande. Detta var inte genomförbart då denna utbildning sker på 50-timmars veckor, och då samtliga studenter hade kandidatarbete under utvecklingsperioden vilket ledde till en tung total arbetsbelastning. I efterhand hade Gemensam kodstandard säkerligen varit ett bra val, för att förenkla för samtliga utvecklare att sätta sig in i varandras kod och för att skapa en renare kod. Det användes dock inte då införandet av en sådan inte var ett prioritetsbeslut och då medlemmarna inte förutsåg att kodstandarden skulle komma att bli varierande. Detta orsakade inte några större problem i detta relativt lilla och enkla projekt, men i ett framtida projekt som möjligtvis skulle varit större skulle en gemensam kodstandard vara att föredra, särskilt då utvecklingen sker parallellt.

Brainstorming

För de val som uppkommit under projektets gång, särskilt gällande vilken applikation som skall utvecklas, samt vissa beslut kring layout och funktionalitet så har gruppen använt sig av brainstorming. Detta innebär sessioner där alla gruppmedlemmar får vädra sina idéer utan negativ kritik och att beslut så småningom fattas genom röstning. Detta har varit ett effektivt och passande verktyg för att alla medlemmar skall få chansen att uttrycka sina förslag och för att göra beslut så rättvisa som möjligt. Det är även ett bra verktyg för att komma på kreativa lösningar då nya idéer ofta triggas från de förslag som uppkommer under dessa sessioner. Det är passande vid tillfällen där kreativitet är viktigt och mindre användbart då specifika lösningar skall tas fram som kräver spetskompetens. I dessa fall finns risken att röstningen skulle kunna leda till fattandet av mindre optimala beslut, särskilt om alla medlemmar inte har nödvändig kompetens för att fatta goda beslut gällande ämnet. I sådana fall kan det vara bättre att beslut fattas av en person med särskild kompetens alternativt att röstningen viktas eller är unison.

Scrum:

Scrum är en agil mjukvaruutvecklingsmetod som är iterativ och inkrementell. Syftet med metoden är att skapa en flexibel och holistisk utvecklingsstrategi där utvecklingsteamet arbetar som en enhet för

att nå ett gemensamt mål. En viktig princip gällande Scrum är tillkännagivandet av faktumet att en kund kan ändra sina önskemål på produkten under utvecklingens gång. Dessa oväntade förändringar är svåra att hantera i mer traditionella metoder där den mesta planering görs och de flesta beslut fattas i projektets startskede. Att i dessa fall ändra specifikationer efterhand kan ge förödande konsekvenser och innebära stor omarbetning.

Scrum innebär därför en så kallad ”empirical approach”, vilket är ett förhållningssätt som betyder en acceptans av att ett problem inte kan definieras eller lösas på förhand, vilket kräver att ett team måste kunna hantera nya krav och vara snabba på att leverera. I Scrum är det, likt i eXtreme Programming, viktigt med nära kommunikation och samarbete mellan utvecklarna, med skillnaden att i Scrum sker utvecklingen vanligen parallellt, även om fysiskt tillsammans. I Scrum fungerar det även med online-samarbete så länge kommunikation sker ofta, helst dagligen.

Det finns i huvudsak tre viktiga roller i Scrum-metodiken nämligen*: Scrum Master, Product Owner* samt *Development team*. Scrum Mastern är en slags ”coach” för teamet som har som uppgifter att synkronisera olika aktörer, upprätthålla att processen går rätt till samt lösa problem som hindrar utvecklingsgruppen. Product Owner, eller produktägaren, hanterar och prioriterar krav och önskemål på produkten och representerar kunden. Denne har ansvar för att se till att utvecklingen genererar värde affärsmässigt. Både dessa ovanstående kan vara medlemmar av utvecklingsteamet men de får ej vara samma person. Utvecklingsteamet är utvecklarna av koden vilka skall vara självorganiserande och gärna tvärfunktionellt utformad för att få med all nödvändig kompetens.

En viktig beståndsdel av Scrum-metodiken är att jobba i iterativa så kallade ”*sprints”.* Detta innebär en viss tidsperiod, vanligtvis mellan en vecka och en månad, där en delprocess genomförs som sedan upprepas i nästa sprint. Denna delprocess startar med ett planeringsmöte där uppgifterna som skall utföras i sprinten identifieras och avslutas med ett utvärderande möte där sprintens framgång analyseras och lärdomar tas med till nästa sprint. En uppgift som utförs i sprinten skall ha gått igenom en design-code-test cykel för att anses vara färdig.

En annan viktig princip i Scrum är möten. Två huvudsakliga möten beskrevs ovan, nämligen planering och utvärderingsmöten för varje sprint, men Scrum innefattar även så kallade ”*daily-scrum meetings”* som görs för att gruppmedlemmarna skall stämma av framgången för deluppgifterna och vädra eventuella problem.

Scrum utgörs även av diverse *artifacts,* beståndsdelar*,* såsom product backlog, sprint backlog, burndown charts och product increment. Den förstnämnda innebär en sammanställning av samtliga önskemål och även prioriteringen av dessa. Sprint Backlog är den delen av Product Backlog som innebär det som gruppen åtar sig inför varje sprint. Product increment, eller inkrementet, är den körbara produkten som existerar efter en sprint, det vill säga det som gjorts fram tills den tidpunkten. Burndown chart är en graf som visar det arbete som är kvar i Product Backlog.

I detta projekt eftersträvades användningen av en del principer i Scrum. Anledningen till det är framförallt då det ansågs vara en passande metod för projektet samt då gruppen ville pröva metoden i lärande syfte. Vissa delar av metodiken valdes dock bort på grund av olika anledningar, exempelvis valdes daily-scrum-meetings bort på grund av svårigheter att fysiskt träffas samt på grund av tidsbrist. Andra delar som valdes bort var vissa artifacts, såsom burn-down charts, utav tidsbrist och då det inte kändes som de hade tillfört så mycket.

De aspekter av Scrum som användes i projektet var framförallt parallellprogrammering. Detta på grund av att det var lättare och mer effektivt att arbeta på detta sätt, särskilt då vi hade svårigheter att få ihop våra scheman. Vi använde även ett kundorienterat arbetssätt, med utformandet av user-stories, och i detta fall var Product Owner hela gruppen och inte en enda person och alla hade därför rättighet att lägga till och ändra user stories. Vi använde även en del dokumentation, särskilt Product Backlog. Tanken var att ha två-veckors-sprints där uppgifter delegerades ut i form av user-stories, men det var inte alltid som dessa två-veckors-sprints höll. Det brast även vad gällde planerings- och utvärderingsmöten (på grund av olika samordningsproblem) vilket inte var optimalt då det drog ner effektiviteten hos arbetssättet.

Fördelar med denna metod är, om korrekt utförd, att det blir tydliga delmål och det blir kontroll över tidsaspekten för uppfyllandet av dessa delmål, vilket gör att arbete inte skjuts upp eller liknande. Det är även lätt att följa processen tack vare den rikliga mängd dokumentation som används, och det är lätt att se vad som gjorts i respektive period. Det är även bra att kunden tas hänsyn till genom hela processen med hjälp av user stories och en Product Owner som har ansvar för att föra kundens önskan. Det är även positivt att ha såpass god uppföljning, i och med möten och kommunikation, då det främjar lärandet, koordinationen av arbetet och skapar kontinuerlig förbättring. Det är även bra att det finns utrymme för delegering och ansvarsfördelning då utvecklingen ofta sker parallellt och individuellt.

Nackdelar med Scrum är att det är en tidskrävande metod, speciellt om korrekt utförd. Alla möten och dokumentering som krävs tar tid från själva utvecklandet och det tar även tid att sätta sig in i metodiken ordentligt om man inte har erfarenhet från den sedan tidigare. I enklare projekt kan vissa aspekter ses som lite onödiga, även om det kan vara extremt viktigt i större, mer komplexa projekt. Om metoden utförs korrekt kan det också finnas nackdelar då det lätt blir mycket ”micro-management” då Product Owner har såpass mycket makt över vad som skall utvecklas och när, speciellt i och med dagliga uppföljningsmöten. Risken kan då vara att utvecklarna får för litet utrymme för kreativitet och eget ansvar, vilket kan ha negativa konsekvenser på trivsel och motivation, beroende på individerna i fråga.

I detta projekt följdes som tidigare nämnt Scrum-metoden inte fullt ut, utav diverse anledningar, men mest på grund av tidsbrist. Det som fungerade bra var framförallt användandet av user stories då det skapade en tydlig bild av vad produkten förväntades uppfylla. Det var ett lätt sätt att hålla koll på vilka delproblem som var prioritet. Att arbeta parallellt fungerade relativt bra och var effektivt för det mesta, särskilt vid de tillfällen kommunikationen var god medlemmarna emellan. Vid vissa tillfällen brast kommunikationen något vilket skapade ovisshet kring vem som skulle göra vad och vid vilken tid. Detta var också en konsekvens av att Scrum-principen med uppföljningsmöten inte följdes i full mån. Detta gjorde att vissa uppgifter tog längre tid än önskvärt och gjorde processen mer ineffektiv.

I ett framtida projekt hade det fungerat bättre ifall kommunikationen och uppföljningen hade varit mer noggrann. En annan brist i detta projekt som hade varit av vikt att förbättra hade varit att ha en enda fysisk person som var Product Owner istället för att hela gruppen var det. Detta hade förenklat arbetsfördelningen och uppföljningen om en specifik person hade fått ansvaret att se till att processen gick rätt till. I detta projekt blev det vissa problem då medlemmarna hade något skiftande uppfattningar om vad som var viktigast att göra först och hur lång tid uppgifter fick ta. Utan en tydlig

ledare kunde det vara svårt att avgöra dessa konflikter utan att orsaka dålig stämning. För övrigt fungerade metoden bra och den skulle ha fungerat bättre ifall vissa justeringar hade gjorts och ifall Scrum-principerna hade följts mer utförligt.

Denna metod lämpar sig bra när det finns en tydlig kund som produkten skall anpassas till samt då utvecklarna är tillräckligt erfarna i området för att kunna programmera på egen hand. Vill man ha högt tempo på kodutvecklingen är denna metod bättre än parprogrammering. Är det mer otydliga uppgifter, då det är svårare att veta vart arbetet skall börja, kan det vara mer problematiskt att arbeta på detta sätt. Ifall kodkvalitet och testning är av högsta prioritet är det viktigt att alla utvecklare har hög kompetens inom detta samt att utvecklingsgruppen inte består av för många medlemmar då detta skulle kunna skapa integrationsproblem.

Testning:

De tester som användes i detta projekt var främst manuella Black-Box-testing samt vissa JUnit-tester. JUnit-testerna användes för att testa att databasen fungerade korrekt då detta var en central del av projektet. För övrig funktionalitet användes manuella Black Box-tester för att se till att user stories uppfylldes på ett tillfredställande sätt. Anledningen till att övrig testning, såsom vissa typer av White Box-testning inte användes i någon större utsträckning var framförallt på grund av tidsbrist och även viss bristande erfarenhet kring vissa typer av automatisk testning. Då denna produkt ändå utvecklades med kunden och användaren i fokus, så ansågs det ändå allra viktigast att samtliga user stories uppfylldes, vilket kunde testas med diverse manuella Black Box-tester.

Ifall projektet hade fortgått skulle fler JUnit-tester ha kunnat utformas och fler utförligare tester kunnat göras, särskilt med avseende på White-Box testing.

GitHub:

För versionshantering i projektet användes GitHub. Det tog ett tag att sätta sig in i detta verktyg samt att förstå hur det bäst skulle utnyttjas då inga gruppmedlemmar hade använt det tidigare. I början av projektet uppkom en del problem då medlemmar arbetade på delar som inte ”pushades” upp i direkt anslutning till att en uppgift var klar. Då kunde det bli en del problem ifall väldigt mycket funktionalitet hade lagts till utan att övriga medlemmar fått ta del utav de nya ändringarna. Detta gjorde både att övriga medlemmar inte kunde se vad som hade gjorts, vilket är av vikt då en grupp har begränsade fysiska möten, och att vissa merge-konflikter uppstod som behövde lösas manuellt. I detta projekt var dessa konflikter av enklare karaktär och kunde därför lösas relativt snabbt, men av detta drogs lärdomen att vara noga med att ”pull” den senaste versionen innan ny kod skulle utvecklas, samt att ”push” så ofta och så fort som möjligt. I resterande delar av utvecklingen fungerade användningen av GitHub bra, då det möjliggjorde ett enklare sätt att hålla kolla på vad som gjordes på olika håll i gruppen. Detta var särskilt användbart när medlemmarna hade svårt att träffas och följa upp vilka framsteg som gjorts.

Utvärdering av projektet

Det som fungerade bra i projektet var användningen av user stories vilket skapade en god överblick på vad som behövde göras. Den inledande fasen av projektet var också bra eftersom mer uppföljning gjordes och fler möten hölls. Den avslutande fasen av projektet var också bättre då arbetet flöt på mer effektivt och fler uppgifter slutfördes. Detta berodde delvis på grund av att medlemmarna hade

mer tid över i dessa faser på grund av en lättare arbetsbörda i pågående kandidatarbeten. Användningen av GitHub fungerade också bra och det var bra att uppgifter delegerades mellan gruppmedlemmarna även om det fanns vissa brister gällande arbetsfördelningen, kommunikationen och uppföljningen.

Det som alltså fungerade mindre bra var kommunikationen i vissa delar av projektet, särskilt i den mellersta fasen. Det blev viss ojämn arbetsfördelning i gruppen. Som tidigare nämnt blev det vissa problem på grund av att det inte fanns någon tydlig Product Owner och på grund av dålig uppföljning mellan sprinterna och för få möten. Ibland utfördes vissa uppgifter inte inom rätt sprint vilket skapade viss problematik.

Vad gäller gruppsamarbetet fungerade det okej i vissa delar av projektet särskilt i början och även i slutet. I mitten av projektet fungerade det mindre bra på grund av svårigheter att ses och på grund av att medlemmarna hade olika tidsfönster för inlämning av kandidatarbeten. Detta gjorde att vissa medlemmar inte kunde arbeta under vissa tidsperioder och resterande medlemmar inte kunde arbeta under andra perioder. Detta hade kunnat fungera bättre om arbetet har planerats på ett sätt där medlemmarna turades om gällande arbetsbördan. Medlemmarna hade också varierande preferenser för arbetssätt och även varierande ambitionsnivå i vissa perioder vilket ledde till vissa svårigheter. Dessa problem hade säkerligen blivit mindre påtagliga ifall uppföljning och kommunikation varit bättre.

De lärdomar som dragits av projektet och de ändringar som skulle ha gjorts till ett framtida projekt rör framförallt arbetssättet och användningen av Scrum. Som tidigare nämnts hade Scrum-principerna för uppföljning och möten följts mer noggrant, samt så hade en enda Product Owner utsetts. På så sätt hade processen varit mer effektiv och delmål hade uppfyllts inom rätt tidsram vilket hade minskat en del stress mot slutet av projektet. Arbetsfördelningen hade gjorts på ett jämnare och mer optimalt vis. Mer noggrannare testning hade gjorts ifall tid och resurser funnits samt att dokumentation hade förts mer kontinuerligt under arbetets gång.

Gällande själva koden och applikationen är gruppmedlemmarna relativt nöjda med utfallet då de flesta user stories blivit uppfyllda. Inga större ändringar hade gjorts i upplägget, designen eller kodstrukturen ifall projektet hade gjorts igen och gruppen anser att applikationen är utformad på ett vis som möjliggör justeringar i funktionalitet och krav. Detta gäller särskilt databasens upplägg då dess utformning tillåter att tabeller ändras, läggs till, används, uppdateras och tas bort på ett enkelt sätt, vilket gör att det är lätt att justera efterhand vilken data som skall lagras och på vilket sätt. Däremot hade en bättre och mer effektiv process gjort att applikationen hunnit få fler funktioner, exempelvis funktionen med anslutning till grupper, vilket var ett initialt önskemål. Visionen som gruppen hade specificerat blev dock uppfylld vilket var det absolut viktigaste kravet och slutprodukten måste därför anses vara lyckad, trots vissa brister i framtagningen av denna.