UGGLETRÄSKET

Chalmers Tekniska Högskola
2015-05-27

Alexander Fastberg
David Kron
Daniel Larsson
Minh Bang Nguyen
Johan Nordholts
Niklas Näkne

Inledning

Visionen som låg till grund för detta projekt är som följer:

“Vår vision är att skapa en användarvänlig applikation som lärare kan använda för att skapa och dela ämnesspecifika frågor till såväl elever som andra lärare.”

Valet av vision grundar sig i att projektets medlemmar som studenter kan relatera till den problematik som stundtals uppstår under studietiden. Under vissa perioder kan kunskapsinlärning kännas tungt och tråkig och då skulle roligare alternativ vara välkomna. Projektet är således vårt bidrag till att skapa en roligare läromiljö för studenter såsom för lärare. Genom att lärare kan skapa specifika frågor och frågelistor kan de göra innehållet anpassat för sina studenter. Bra innehåll kan sedan spridas publikt eller hållas privat om frågorna är specifikt inriktade på exempelvis en klass eller om man som lärare vill att studenterna ska besvara frågor direkt i klassrummet.

Den applikation som visionen beskriver är således dynamisk genom att användaren kan använda grundläggande funktionalitet på olika sätt. Hur som helst kändes visionen för bred för vårt projekt att uppfylla i sin helhet, vilket är anledningen till att den produkt som vi för tillfället har erbjuder grundläggande funktionalitet som att skapa och besvara frågor. Därför finns ett visst behov av vidareutveckling, något som kommer ske efter att detta projekt är avslutat. Eftersom innehållet och stora delar av värdet som denna applikation ska bidra med skapas av användarna, kan den produkt som presenteras bildligt ses som ett tomt skal som i framtiden fylls med mjölk och honung.

Projektupplägg

Projektet genomfördes enligt en scrummetodik, där olika funktioner skapades parallellt. Utvecklingsteamet på sex personer delades inför varje sprint upp i teams. Varje sprint pågick under en veckas tid och avslutades med en avstämning med kunden. Regelbunden kontakt med kund är en byggsten inom agil programutveckling och viktigt för att säkerhetställa att produkten, i det här fallet applikationen, utformas enligt kundens önskemål. Under mötena med kunden presenterades vilka funktioner som adderats under veckan samt planer inför nästkommande veckas sprint. Totalt bestod projektet av fem sprintar.

Planering och koordination

Som följd av att samtliga i projektet var i slutfasen av respektive kandidatarbete krävdes planering och koordinering för att få tiden att gå ihop. Planeringen gällde framförallt scrummöten och därför initierades innan projektets start ett tavelmöte, där de inblandade medlemmarna fick visualisera sitt schema för resterande del av gruppen. Detta gjordes i syfte att hitta gemensamma tider som främst scrummöten kunde hållas på, vilket var utmanande då schemat för fyra olika kandidatgrupper behövde beaktas. Två tider lyckades sedan identifieras. För att underlätta koordinering upprättades en loggbok i den inledande delen av projektet. Här noterades detaljer om alla scrummöten där exempelvis utvärderingar och kommentarer av sprintar ingick. De valda metoderna anses vara lyckade eftersom att schemat följdes genom hela projektet vilket resulterade i att varje individ i god tid kunde planera sin tidsdisponering under kommande sprintar. Vidare har arbetet med loggboken resulterat i ett kontinuerligt förbättringsarbete vid utformning av sprintar.

Till en början var tillämpningen av scrum och att skapa funktioner parallellt svårt att få till. Det berodde på att grundfunktioner så som databasanslutning var tvungna att finnas innan det fanns möjlighet att dela upp arbetet. Därförförsökte vi exempelvis dela upp arbetet så att vissa tittade på applikationens övergripande struktur, vissa satte sig bättre in i hur olika tekniska verktyg fungerar medan vissa undersökte möjligheter till en smidig databaslösning.

I efterhand anses denna metodik ha varit ineffektiv, och hade vi startat projektet med den kunskap och erfarenhet vi besitter idag hade vi antagligen inte försökt att arbeta med parallella områden redan från första sprinten. Det hade istället varit önskvärt om vi på något sätt kunnat möjliggöra att samtliga arbetade på grundfunktionerna tillsammans i sprint ett eftersom de är vitala för resten av arbetet. Därefter skulle man kunna påbörja sprint två med jämlika kunskaper gällande samtliga grundfunktioner och då kunna fördela uppgifterna på ett bra sätt. Så fort grundfunktionerna var implementerade blev förutsättningarna bättre och arbetet enligt scrum började fungera väldigt bra. Så fort vi kunde börja fördela ut tydliga arbetsuppgifter över teamen märktes det att gruppens arbetspotential utnyttjades på ett bättre sätt och projektet tog fart.

Varje sprint innehöll två schemalagda möten inom utvecklingsteamet, varav ett möte hade som syfte att avsluta och utvärdera föregående sprint samt påbörja och planera nästkommande sprint. Det mötet hölls under samtliga veckor direkt efter avstämningsmötet med kunden. Anledningen till att mötet placerades efter kundmötet var att ha kundens tankar färskt i minnet och utifrån det kunna på ett bra sätt utvärdera föregående sprint samt planera nästa. Det andra schemalagda mötet låg i mitten av varje sprint med syfte att kontrollera att samtliga teams hade kommit igång och uppfattat sina arbetsuppgifter på ett korrekt sätt. Mötet i mitten av varje sprint var också ett bra tillfälle att ta upp eventuella problem och ta hjälp av övriga teams för förslag på lösningar. Överlag fungerade dessa schemalagda möten väldigt bra vilket är något vi tar med oss till framtida projekt. En nyckelfaktor för projektet var sprintmötena direkt efter möte med kund då det möjliggjorde att samtliga i utvecklingsteamet kunde vara på plats samtidigt.

Arbetsfördelning

För att försöka få en jämn arbetsfördelning inom utvecklingsteamet gjordes en fördelning baserat på user stories. Verktyget effort points tillämpades under tre av fem sprintar och innebär att olika uppgifter värderas utifrån beräknad arbetsinsats relativt varandra. I projektet uppskattades arbetsinsatsen utifrån antalet arbetstimmar, vilket valdes då det upplevdes som enklast för utvecklingsteamet att förhålla sig till. Det användes ingen statisk graderingsskala, utan graderingen skedde genom att betrakta tidsåtgången för föregående sprint. Det resulterade i att en effort point inte symboliserade lika mycket arbete för första respektive sista sprinten. När uppskattning av effort points gjordes togs även hänsyn till teammedlemmarnas erfarenhet och kunskap.

Fördelen med effort points är att man på förhand skapar sig en förståelse för problemets omfång. Det blir således enklare att uppskatta hur mycket tid som behövs för att genomföra en viss user story, vilket leder till bättre optimering av tiden som finns inom en sprint. Det kräver dock viss förståelse för projektet i allmänhet och en user story i synnerhet då det är svårt att förutspå handlingar som man saknar kunskap om. Detta blev påtagligt i vårt projekt, framförallt i de inledande sprintarna där effort points tillämpades. Detta var ett resultat av att utvecklingsteamet hade begränsade kunskaper om främst Android Studio och Git, vilket gjorde det svårt att förutspå hur mycket tid som skulle läggas på tekniska respektive programmeringsrelaterade problem.

Prognoserna för arbetsåtgången blev bättre ju längre projektet fortskred. Vi insåg att effort points skulle bidra med mer i ett längre projekt eftersom uppstartsfasen med att lära sig bedöma arbetet har en viss startsträcka. I ett kort projekt som detta blev startsträckan oproportionellt lång vilket resulterade i att de positiva effekterna som effort points ämnar bidra med uteblev. Det bidrog dock till förståelse för hur metoden kunde appliceras vilket var en viktigt lärdom vi tog med oss.

Testning och verifiering

Intern testning har utförts på i princip samma sätt oavsett om testningen har behandlat databaser, Java-kod eller XML-kod. För att säkerställa att de förändringar som gjorts bibehåller och förbättrar existerande funktioner har en form av kontinuerlig testning gjorts. I praktiken innebär det att små förändringar görs och att dessa sedan säkerställs genom att programmeraren som utfört förändringen testar den manuellt. Vid större förändringar har flera programmerare testat funktionen genom att manuella tester gjorts på flera olika enheter, exempelvis emulatorer och fysiska mobiltelefoner. En fördel med detta förfarande är att de tänkta användarna kan ha olika sorters enheter och olika versioner av mobilt operativsystem, vilket ställer högre krav på applikationens anpassningsbarhet. Detta sköts nästan uteslutande av Android Studio, men behovet att testa på olika enheter har varit framträdande när det kommer till designval eftersom dessa ofta påverkas av hur den fysiska enheten ser ut. Det har således inte tagits in en separat testare och den stora anledningen till det är att programmeraren är bäst lämpad för att förstå och åtgärda de fel som kan tänkas uppstå. Detta gäller framförallt då det det mestadels handlar om sådant som snabbt går att ordna. Utöver intern testning har även kunden i viss mån testat applikationen vid de veckovisa mötena vilket givit inspirerande input gällande funktionalitet och design.

IT-relaterade verktyg

De huvudsakliga IT-relaterade verktyg som använts har varit Git, GitHub, Android Studio och PHP MyAdmin. Även Google docs har använts för att dokumentera arbetsprocessen och främst för att sammanställa denna post mortem rapport. Android Studio är den plattform som användes för att koda själva androidapplikationen. Ett alternativ som det inledningsvis funderades på att använda var Eclipse som kan tillhandahålla samma verktyg genom diverse plug-ins. Trots tidigare erfarenhet av Eclipse inom utvecklingsteamet valdes detta alternativ bort till förmån för Android Studio. Anledningen till detta var att Android Studio var enklare att använda, vilket prioriterades då gruppens medlemmar saknade erfarenhet av att programmera androidapplikationer. Vi upplevde inte att det fanns någon begränsning hos Android Studio varvid detta verktyg skulle vara lämpligt i framtiden, även vid större projekt.

Git

Git och Github användes för att kunna dela kod och hålla ordning på olika versioner av applikationen. Det användes även för att möjliggöra att flera användare, på ett enkelt och smidigt sätt, skulle kunna arbeta parallellt med projektet. Det finns andra verktyg som fyller samma funktion som en versionshanterare och ett av dessa är Trello, vilket beaktades i inledningen av projektet. Detta verktyg är ett mer grafiskt anpassat verktyg i jämförelse med Git som använder kommandotolken. Detta kändes som ett rimligt val med tanke på att medlemmarna i projektet inte var vana vid att använda just kommandotolken. Ett annat motiv bakom användandet av Trello var att en av gruppmedlemmarna hade använt det i ett tidigare projekt.

Med grund i detta testades Trello. Implementeringen gav dock upphov till en del problematik som skulle vara var tidskrävande att lösa. Då projektet är relativt kort togs beslutet att istället använda Git och kommandotolken eftersom denna lösning fungerade. Även om Git kräver en viss förståelse innan det kan börja användas fungerade det bra när allt var på plats och samtliga medlemmar förstod de grundläggande funktionerna. En viss lägstanivå krävdes för att kunna använda vektyget, vilket resulterade i att mer tid lades på att förstå verktyget Git i början av projektet. Den extra tid som verktyget tog upp relativt den tid som lades på att koda minskade med arbetets gång och utgjorde en väldigt liten del av slutfasen.

Git gör det möjligt att arbeta i olika versioner på ett projekt för att sedan slå ihop de olika lösningarna till en master-version. En version ligger i en branch som är tillgänglig för alla medlemmar i projektet. I inledningen av projektet arbetade alla medlemmar på samma version, alltså i masterbranchen. Efter hand tilldelades istället varje team i projektet en egen branch där de kunde arbeta och de olika versionerna kombinerades sedan med masterbranchen i slutet av varje sprint. Varje team kunde sedan ladda ner senaste master-versionen att arbeta med i kommande sprint. Beroende på hur nära de funktioner som de olika teamen arbetade på låg varandra kunna vissa mergekonflikter uppstå i samband med merge mot master. Dessa behandlades genom att en person var ansvarig för mergen av samtliga branches mot master. Denna person tog sedan hjälp av en person från det team som var ansvarigt för den branch som resulterade i en mergekonflikt. På så sätt löstes konfliken och rätt kod valdes. Arbetssättet fungerade bra eftersom den person som oftast var ansvarig för mergen hade en god insikt i samtliga delar av projektet. Det kan tänkas att det i stora projekt kräver mer koordinering för att lyckas genomföra en god merge, något som inte var några problem i vårt relativt lilla projekt.

I framtida projekt borde Git användas i samtliga fall och det bidrar speciellt mycket i de fall där många programmerare är involverade. Det skulle även vara användbart vid mer omfattande individuella projekt eftersom det bidrar till att hålla ordning på olika versioner. Då kan ändringar göras och man kan samtidigt gå tillbaka till en tidigare version om det visar sig att en viss funktion inte är tillräckligt bra. Vid ett mycket litet individuellt projekt där man i förhand har ett begränsat omfång skulle dock Git kännas överflödigt.

Github användes som samlingsplats för filerna som användes i projektet. Valet att använda Github som är en publik samlingsplats valdes eftersom detta projekt var ett studierelaterat projekt. Det användes även på grund av den goda dokumentation på internet som finns om denna plattform. Skulle ett projekt göras med tanke att kommersialisera produkten skulle dock inte en publik samlingsplats vara att rekommendera. I ett sådant fall skulle i stället ett privat samlingsställe användas.

Databaser

En stor del av arbetsbördan lades på att finna en fungerande lösning som kunde tillhandahålla en databas, vilket kan göras på flera sätt. Det vanligaste lösningen i stora projekt är att ha egna servrar som programmet eller applikationen arbetar mot. Fördelen är då att man kan konfigurera dessa servrar efter önskemål, men då ställs krav på underhåll och förståelse för hur hårdvara kring servrar fungerar. Eftersom ingen i utvecklingsteamet besitter dessa kunskaper valdes detta alternativ bort. Det finns också leverantörer som kan tillhandahålla gratis molnbaserad lagring av data, men datamängden hos dessa är ofta reducerad och tillförlitligheten låg. I vårt fall fanns inga medel för att finansiera datalagring varvid vi var tvungna att hitta en gratislösning. Detta tog inledningsvis mycket tid eftersom att vi först var tvungna att sätta oss in i vilken funktionalitet som behövdes för vårt ändamål samt att identifiera olika lösningar. Att sedan implementera lösningen var ofta inte heller enkelt vilket resulterade i att flera lösningar testades innan vi bestämde oss för vad som blev bäst.

Vi bestämde oss för att använda en gratisleverantör som som tillhandahåller ett tillräckligt stort minnesutrymme för projektet. Dessutom erbjuds möjligheten till att skapa MySQL-databaser med stöd för phpMyAdmin. phpMyAdmin är ett verktyg för att hantera databashanteraren MySQL via en webbläsare. För att koppla upp applikationen mot databasen användes PHP-filer med innehållande SQL-kommandon, vilka laddades upp på leverantörens servrar. I den databaskurs som utvecklingsteamet tidigare läst användes API:n JDBC för att interagera med databasen. Detta vore en alternativ lösning till PHP-filerna i projektet, men JDBC upplevdes som en väldigt bra lösning då SQL-kommandon kunde skrivas direkt i Java-koden. Anledningen till att den lösningen inte används kan kort och ärligt beskrivas som brist på kunskap kring hur en sådan kontakt upprättas. Avvägningen gjordes att den valda lösningen var fullgod för applikationsområdet. En tydlig nackdel med PHP-filerna är dock att de lätt kan öka i antal om man inte skapar dem väldigt dynamiskt med många inparametrar.

Skulle ett större projekt genomföras, speciellt om man har som tanke att kommersialisera produkten, skulle en egen server, eller ett mer seriös molntjänst användas för datalagring. På grund av projektets begränsade omfattning ansågs den ad hoc lösning som togs fram tillräcklig.

Men om projektet önskas skalas upp behövs en annan lösning och närmast till hands är då att använda en mer seriös moln-baserad tjänst.

Kunskapsanskaffning

De olika metoderna och verktygen som nyttjats i projektet gav överlag upphov till en inledningsvis omfattande inlärningsprocess. Vid projektets start låg alla medlemmar i utvecklingsteamet på olika nivåer gällande kunskap relaterat till ämnet. Under projektets fortskridande utvecklades inlärningsarbetet till en mer kontinuerlig process där kunskapsanskaffning skedde naturligt vid behov.

Programmering

Gällande programmeringskunskaper var den initiala nivån bland medlemmarna i utvecklingsteamet någorlunda jämn då alla sedan tidigare hade liknande erfarenheter av såväl Java som XML. Nivån ansågs vara tillräckligt god för att för att inte kräva en inledande kraftansträngning i den inledande fasen av applikationsutvecklingen. Kunskapsnivån låg även till grund för möjligheten att införskaffa kunskap relaterad till programmering kontinuerligt vid behov. Exempelvis gjordes sökningar efter befintliga Java- och Android-bibliotek då specifika funktioner och implementationsmöjligheter eftersöktes. Dessutom utnyttjades diverse forum på internet i problemlösnings- och felsökningssyfte.

Slutligen anses metodiken att inkrementellt införskaffa kunskap ha varit positiv. Utvecklingsteamet känner inte att det alternativa tillvägagångssättet, med en mer ingående kunskapsinhämtning i början av projektet, hade gynnat arbetet. Istället skulle detta ha lett till ett mer ineffektivt arbete vilket skulle ge ett markant negativt utslag på resultatet av projektet, sett till den snäva tidsramen.

Databashantering

Det påtagliga inslaget av databashantering i projektet har medfört att även kunskap om PHP och SQL krävts. Kunskaper om SQL har alla projektmedlemmar kunnat inhämta i tidigare kurser. Dessa har varit fullgoda för projektets syfte och spås också kunna bidra till en vidare utveckling av applikationen. Däremot har kunskapsanskaffning relaterat till PHP varit nödvändigt. Detta var framförallt tydligt i den inledande fasen av projektet då de mest grundläggande funktionerna implementerades. Därför krävdes en relativt omfattande inlärningsprocess av PHP tidigt. Då majoriteten av projektmedlemmarna aldrig tidigare arbetat med programmeringsspråket blev det naturligt att de med erfarenhet hjälpte övriga i teamet. Med anledning av den något korta tidsramen för projektet ansågs det vara orimligt att behöva vänta på att alla medlemmar skulle nå den kunskapsnivå inom PHP som krävdes för applikationen. Det medförde i sin tur att de med tidigare erfarenhet av programmeringsspråket fick ta på sig en stor arbetsbörda tidigt i projektet. Anledningen är att teamet ville undvika att ha databasimplementationen som det stora hindret i början av applikationsutvecklingen. Detta gav i slutändan ett positivt resultat i en tillfredsställande implementering av databaslösningen, även om arbetsbördan av denna implementation blev snedfördelad.

Verktyg

Utnyttjandet av Android Studio som utvecklingsmiljö var nytt för alla i utvecklingsteamet med undantag för en medlem. Av denna anledning krävdes en initial tidsperiod för de flesta att bekanta sig med miljön. Detta sågs dock aldrig som ett bekymmer då miljön tycktes vara relativt logisk och användarvänlig i relation till andra utvecklingsmiljöer som utvecklingsteamet tidigare arbetat i, däribland Eclipse. Därför behövde den projektmedlem som tidigare haft erfarenhet av Android Studio bara avsätta en relativt liten andel tid med att hjälpa övriga i teamet.

Däremot erfordrades en betydligt större kraftansträngning för att komma igång med versionhanteraren Git. Detta var det absolut mest krävande verktyget att börja nyttja, både i termer av arbete och tid. Likt Android Studio var det bara en i teamet som tidigare hade erfarenhet av Git. Därför avsattes en stor del av dennes tid för att försöka minska kunskapsgapet till övriga. Detta medförde att en betydande del av projektet ägnades åt implementering, exempelvis installera mjukvara på respektive medlems dator, men framförallt förstå verktygets olika funktioner.

I slutändan blev resultatet positivt då alla i utvecklingsteamet anser att Git var oerhört givande eftersom det möjliggjort en uppdelning av arbetet i samband med en tydlig versionshantering. Alla är dock överens om att implementationsfasen hade kunnat vara betydligt mer effektiv om alla i utvecklingsteamet hade en bättre kunskapsgrund att stå på, istället för att låta en stor del av arbetet tillfalla färre medlemmar. Därför är det motiverat med en betoning på hur viktigt Git är som verktyg för projektet, för att arbetet med implementering av verktyget ska ske så tidigt som möjligt.

Sett till valet kring metodiken Scrum hämtades den kunskapen från föreläsningarna och den specifika övningen med LEGO® gällande Scrum. Vidare utveckling av arbetet med Scrum skedde i samråd med handledare veckovis, där gamla arbetsmetoder kompletterades och korrigerades genom handledningen. Gästföreläsningen med Spotify uppmuntrade till rotation mellan teamen vilket testades i projektet. I övrigt användes internet mycket för att skaffa inspiration till olika lösningar samt för felsökning.

Teams

Vid projektets scrummöten delades projektgruppen in i mindre teams för att genomföra de user stories som tagits fram under mötet. Storleken på dessa teams bestämdes utifrån den uppskattade storleken på olika user stories. Vilka gruppmedlemmar som ingick i respektive team utgick ifrån gruppmedlemmens tidigare arbetsuppgifter och erfarenheter. Tilldelningen av varje teams arbetsuppgifter baserades på uppskattat mängd arbete, vilken i projektets senare sprintar utgick ifrån arbetet med effort points.

Storleken på grupperna varierade mellan en och tre personer per team. Utformningen av teams utgick till stor del ifrån tanken om *pair programming*, men undantag fanns där större eller mindre user stories bedömdes kräva fler eller färre personer. Arbetet med teams fungerade bra, och förväntas få ännu större fördelar i större projekt, där teams tillåts specialicera sig i högre grad.

Roller

Inledningsvis tilldelades de olika teamen ett arbetsområde i stället för en eller flera specifika user stories. Uppdelningen var att ett team skötte dokumentering, ett team var ansvariga för databashanteringen och ett team för själva kodandet i Android Studio. En sådan uppdelning efter arbetsområde visade sig vara adekvat i projektets inledande stadium, där arbetsuppgifter var diffusa. När en grund lagts för att sedan kunna dela upp projektet lämnades denna uppdelning efter arbetsområde för att istället tilldela varje grupp user stories.

Projektets olika medlemmar tilldelades också roller utifrån de tidiga arbetsområden som identifierats. Dessa roller försvann allt eftersom projektet övergick till att arbeta efter user stories. Senare uppstod emellertid informella roller baserat på medlemmarnas tidigare erfarenheter, där till exempel en medlem fick ta ett stort ansvar för att projektets verktyg fungerade som de skulle för alla medlemmar. Även om projektets rollfördelning inte var särskilt omfattande kunde ändå nytta dras genom medlemmarnas tydliga kompetensområden. Särskilt hjälpsamt var att ha en medlem som hade god överblick över projektets tekniska verktyg. I ett större arbete väntas roller ha en större betydelse, och rolluppdelning kan tänkas vara viktigt om gruppen är löst sammanhållen.

Pair programming

Vid projektets olika programmeringsmoment arbetade varje team i par för att minska risken för fel och öka förståelsen för programmet och koden. I praktiken användes ofta två datorer vid pair-programming. Kodandet genomfördes då på en dator medan den andra datorn användes för att hitta lösningar via exempelvis internetforum, så som stackoverflow. Fördelen med att använda två datorer var att det gick snabbare att implementera och söka efter lösningar. Vid sökning efter lösningar kunde således två datorer användas parallellt vilket reducerade tiden det tog att hitta en god lösning avsevärt. När sedan implementeringen av lösningen skulle göras ökade sannolikheten att

implementeringen gick rätt till. Genom att två personer granskade både lösningsförslaget och implementationen kunde logik-och programmeringsfel enkelt hittas.

Ytterligare en fördel som upptäcktes med pair programming var att det var ett effektivt sätt att dela kunskap. Då teamen byttes varje vecka resulterade det ofta i att de två personer som utgjorde ett specifikt team ofta hade olika stor kunskapsbas. Ofta hade en person bättre förståelse för den delen av projektet som teamet jobbade i och kunde således förklara de ingående delarna för sin arbetspartner på ett bra sätt under arbetets gång. Denne kan i sin tur föra vidare dessa kunskaper vid nästa sprint.

En nackdel med pair programming är att två personer behöver vara närvarande, och att dessa behöver ha någorlunda lika kunskapsnivå. Om kunskapsskillnaden som ovan diskuterats är för stor finns risken att programmeringen bara utförs av den person som besitter den största kunskapen. Problemet uppstod vid ett par tillfällen, och behandlades oftast genom att den person som hade bättre kunskap fick programmera fritt. Den lösningen kan i efterhand starkt ifrågasättas då mycket av de fördelarna med pair programming går förlorade som exempelvis kunskapsspridning och validering.

Ett bättre alternativ hade varit att i stället för att fokusera på att lyckas genomföra en user story, ta extra tid på att dela kunskapen i paret innan implementering av nya funktioner påbörjades. Eftersom projektet bara varade i fem sprintar, totalt fem veckor, var fokus således på att få fram en snabb lösning, vilket också blev fallet. Skulle projektet vara längre skulle inte detta vara hållbart. I ett långt projekt ställas högre krav på att alla ingående personer besitter rätt kunskaper, varvid lösningen med att avsätta mer tid åt kunskapsdelning skulle vara mer motiverad än i ett kort projekt.

Decision making

Beslutsfattning skedde på olika nivåer. Övergripande funktionalitet och design bestämdes vid det veckovisa scrum-mötet genom att user stories definerades och tilldelades varje team. Besluten togs gemensamt av samtliga projektmedlemmar. Det låg sedan på varje teams lott att besluta vad som behövde göras för att förvekliga sina user stories. Denna uppdelning av beslutsfattande visade sig framgångsrik eftersom det är alltför tidskrävande att engagera samtliga medlemmar vid beslut på lägre nivåer. I många fall är det den grundläggande funktionaliteten som är av stor betydelse, och inte mindre detaljer.

Eftersom samtliga inblandade i projektet studerar på samma program och att individerna var väl bekanta med varandra, fördes kontakt kontinuerligt mellan scrummöten. Det resulterade således i att vissa större beslut diskuterades och togs vid andra tillfällen än vid det veckovisa scrummötet. Det nära samarbetet teamen emellan bidrog på ett positivt sätt genom att göra arbetsprocessen dynamisk och medförde att implementeringar och åtgärder kunde genomföras snabbare.

Generell diskussion

Vi arbetade som tidigare nämnts i team men begränsade oss inte av den anledning till att enbart arbeta inom teamet. Sprintarna ansågs ibland vara något korta vilket medför att byten av team och arbetsuppgifter blir kontraproduktivt. Vid längre sprintar kan mer arbete planeras in och den tidsandel som går åt till planering skulle i så fall minskas. Risken att lämna lösa trådar skulle även minska vilket minskar tidsåtgången för överlämnandet av ett specifikt ansvarsområde av projektet. En del arbete skedde över teamgränser som ett resultat av de korta sprintarna. Vi fann trots allt att detta fungerade vilket troligtvis härleds från vår tidigare bekantskap samt projektets begränsande omfattning. Därför användes ingen strikt fördelningsmetod för att fördela teamen vid varje sprint, något som skulle ha behövts vid större och längre projekt. Tidigare bekantskap kan även ses som negativt i det avseende att det finns en existerande dynamik i gruppen, vilken kanske inte är optimalt för projektets utförande. Det är lätt hänt att man hamnar i gamla mönster och arbetar på samma sätt som man alltid gjort. Därför är skapandet av denna post mortem rapport i sig ett värdefullt bidrag.

I ett framtida liknande projekt skulle vi redan från start besitta mycket av den kunskap om tekniker och verktyg som är kritisk för att kunna genomföra ett sådant projekt. Det skulle innebära att vi tidigare skulle kunna komma igång med själva kodandet och skapandet av applikationen. Vi skulle även på ett bättre sätt och i ett tidigare skede kunna använda oss av user stories och dela in arbetsuppgifterna därefter. Vidare skulle vi utnyttja pair-programming mer men även exprimentera med andra sätt att utföra programmering, något som vi känns att vi har haft utrymme till i detta projekt. Pair-programming har används då det schemamässigt varit möjligt vilket varit i en begränsat utsträckning på grund av det parallella kandidatarbetet på pågått under projektets löptid. I stället har en del individbaserad programmering utförts vilket ansetts vara mer tidseffektivt. Dock medför det att validering och verifiering som pair-programming kan bidrag med har nedprioriterats.

Det är svårt att på ett rimligt sätt uppfatta varje individs bidrag till en viss metod. Anledningen är metodvalet som grundas i att projektets medlemmar byter team och uppgifter varje vecka är. Det beror även på att programmering inte bara skett i par utan även individuellt. När tiden här presenteras ingår således både par och individuell programmering i det som betecknas pair-programming, eftersom ambitionen var att använda denna programmeringsmetod. Tidsåtgången är därmed uppdelad efter aktiviteterna; scrummöten, pair-programming samt kunskapsanskaffning. Scrummöten och möten där hela gruppen samlats har i genomsnitt upptagit fem timmar i veckan och den tiden har varit någorlunda konstant genom hela arbetets gång. Pair-programming tog upp mer och mer av den totala tiden som lades på projektet, vilket var ett resultat av att den mängd tid som lades på kunskapanskaffning minskade. Inledningsvis lades cirka tio timmar i veckan på kunskapsanskaffning samtidigt som pair-programming tog drygt fem timmar per vecka. Allt eftersom kunskapsnivån i projektet växte sågs ett skifte av tidsåtgång, vilket

resulterade i att det i slutskedet var mer eller mindre motsatt förhållande. Med andra ord lades 10-15 timmar på pair-programming och uppemot fem timmar på kunskapsanskaffning, beroende på behov.

Genomgående i arbetet belastades arbetsprocessen med det parallella arbetet med kandidaten som samtliga medlemmar var engagerade i, speciellt eftersom kandidatarbetet var i sin slutfas. Detta var olyckligt då inte tillräckligt med önskad tid kunde läggas på detta projekt. Vidare var tidsspannet något begränsat, och det fanns inte tid att experimentera med olika metoder i den utsträckning som önskats. Detta resulterade i att många ad hoc tekniker och lösningar nyttjades, vilket även påvisade att vi i stor utsträckning arbetat resultatorienterat, i stället för att ha fokus på rätt processer. Detta är i sin tur en konsekvens av vår personliga vilja att åstadkomma något konkret och se snabba resultat. Därför användes vissa metoder i mindre utsträckning och stundtals på bristfälligt sätt, vilket vi i efterhand inser är ohållbart i ett större och mer omfattande IT-projekt. Om vi således skulle få göra om detta projekt skulle vi lägga mer tid på att arbete på rätt sätt, eftersom att den kravbild som finns i detta studentprojekt är anpassningsbar. Detta begränsar således inte vilka processer som kan användas.