Post Mortem Report

Processer och metoder

Projektgruppen använde agil iterativ utvecklingsmetodik. Mer specifikt försökte vi följa och använda Scrum som struktur för projektet. Vissa inslag och idéer är dock hämtade från eXtreme Programming. Med försökte menas att vi på bästa möjliga sätt efter rådande omständigheter så som varierande schema och varierande arbetsbelastning från övriga kurser försökte anpassa Scrum till vårt ändamål. Denna kombination av dessa metodiker användes sedan under hela projektet och därför tjänar denna rapport snarare som en redogörelse för kombinationen av Scrum och XP än som en redogörelse för var och en av dem.

Scrum

På grund av situationen utsågs ingen *Product owner* eller *Scrum master* utan alla försökte till viss del utföra dessa uppgifter. Alla hade intressen i den färdiga produkten eftersom alla skulle bli betygsatta utefter den. Därför kan man säga att alla ändå var *pigs,* dvs har en *stake* i projektet.

Till en början sammanställdes en *product backlog* där vi satte upp önskemål för produkten i form av *user stories*. Dessa prioriterades och förfinades allt mer allteftersom de närmade sig ett sådant stadie att de kunde implementeras.

Måndagar tjänade till att avsluta inkrementet(*Sprint review*) och starta en ny *sprint(sprint planning)*. Sprintarna var följaktligen 7 dagar. *Sprint retrospective genomfördes* för att utvärdera Scrum och eventuella ändringar i poängsystem med mera diskuterades. När något togs upp för implementering skulle det vara noga definierat.

Under den första delen av mötet, *sprint review*, diskuterades den nu avslutade sprinten. Här demonstrerades produkten och *product backlog* modifierades för att följa fortskridandet av projektet.

Under den andra delen av mötet, *sprint planning,* gick vi igenom de *user stories* i *product backlog* som vi ansåg var realiserbara och satte poäng på dem efter komplexitet. Av dessa valdes de viktigaste och placerades i *sprint backlog* för implementering under kommande *sprint*.

Scrum meetings genomfördes tre gånger i veckan. Tre dagar och inte fem, som kan ses som standard, valdes eftersom alla skulle ha hunnit med att arbeta uppemot en arbetsdag mellan varje möte. Under dessa korta möten, 10-15 min, gick vi igenom följande frågor.

- Vad har jag gjort sedan förra mötet?
- Vad ska jag åstadkomma till nästa möte?
- Finns det några eventuella hinder?

eXtreme Programming

Från XP hämtades inspiration till våra *user stories.* Dessa användes som uppdrag enligt resonemanget för Scrum ovan. *User stories* poängsattes utefter hur komplexa de var och var och en tilldelades till en ägare innan en *sprint* startades. *Acceptance tests* gjordes för att definiera när en *user story* kunde anses som klar.

Ett kollektivt ägandeskap av koden användes och vi strävade mot att alltid ha en fungerande och väl testad version av applikationen i förvaret. Som en följd av det kollektiva ägandeskapet genomfördes refaktorering och omstrukturering av koden regelbundet. Vi strävade även mot att ha flera *releaser* under perioden.

Tid spenderad

Inkluderat allt arbete i kursen spenderade jag i genomsnitt 20 timmar per vecka. Ambitionen var att hålla en jämn belastning varje vecka, vilket är det optimala med Scrum. Dock lyckades jag inte med det på grund av att jag under tiden gjorde ett kandidatarbete i en annan grupp, vilken inte var så bra på att hålla sin arbetsbelastning lika jämn. Därför blev det mer jobb vissa veckor och mindre jobb andra.

Två timmar i veckan lades på att lära sig veckans material, uppskattningsvis två timmar lades på möten och övrig tid på utveckling och den informationsinsamling som behövdes för att implementera funktionalitet.

Min uppfattning är att även andra medlemmar i gruppen lade ungefär lika mycket tid på kursen. Detta ser jag som en följd av metoden vi använde som var väldigt transparant. Alla kunde se vad de andra gjorde och ingen tilläts halka efter.

Utvärdering av metodik

Detta avsnitt åsyftar till att utvärdera den agila metodik som använts under utvecklandet av applikationen. Som tidigare nämnt är metoden en blandning av Scrum och XP.

Innan projektet hade jag främst erfarenheter att utföra programmeringsuppgifter i andra kurser men även av en annan, mindre lyckad, projektkurs utan någon särskild struktur och utvecklingsmetodik. De första kan liknas vid små vattenfallsprojekt med en klar specifikation innan där slutprodukten sedan valideras av beställaren, rättaren. De andra kan mer liknas vid detta projekt då vi som grupp var beställare och hade en vision för hur projektet skulle arta sig. Eftersom jag har denna erfarenhet är det den jag försöker relatera till.

Fördelar

Den första fördelen och även den största var den kontinuerliga utvecklingen och utvärdering av varje funktion. Den möjliggjorde att vi alltid hade koll på var i projektet vi befann oss och bidrog även till att snabba upp utvecklingen då ingen kunde dölja vad denne hade gjort. Ingen var heller arbetslös då det hela tiden var klart vad som skulle göras. Denna transparants, vilket inte var fallet i mitt tidigare projekt, var väldigt nyttig. Alla kände ansvar.

De kontinuerliga mötena bidrog även till att göra en bättre och mer användbar produkt då nya idéer kom fram under projektets gång och togs tillvara. Idéer som från början ansågs bra kanske inte längre ansågs lika bra när det var dags att lägga till dem. Förändring av user stories och produkten gav helt enkelt konkurrensfördelar för produkten. En lärdom av detta är att det inte alltid står klart hur en funktion skall fungera när den först myntas.

Scrum inbjuder också till personliga möten ansikte mot ansikte vilket gör kommunikationen mycket mer effektiv och precis än till exempel mail som jag tidigare provat. Processen gjorde också alla i gruppen mer involverade och engagerade i produkten vilket bidrog till en bättre produkt.

Jag gillade personligen också att agila processer prioriterar funktionalitet och fungerande programvara före dokumentation och kommenterad kod. Det bidrar till en mer funktionell produkt, snabbare. Det bidrar också till att det bli roligare att jobba med projektet vilket märks i kvaliteten.

Att fokusen ligger på att implementera *user stories* bidrar också till att man får ut så mycket funktion som möjligt på så liten tid som möjligt. En fungerande funktionsrik applikation finns alltid tillgänglig. Pivotal Tracker hjälper på ett illustrativt sätt att åskådliggöra detta.

Nackdelar

De nackdelar med att jobba med Scrum jag har upptäckt under projektets gång beror nog främst på att det är en kurs som ska arbetas med på högskolan tillsammans med andra studenter. Studenter som liksom jag också har andra kurser att jobba med och svårt att hitta gemensamma tider. Det är även svårt att välja ut ett fungerande team så som är möjligt på ett företag. I övrigt har jag svårt att se alltför stora nackdelar med Scrum för projekt på företag.

Varierande schema ledde fram till att medlemmarna jobbade på olika håll. Därför var det svårt för gruppens medlemmar att hålla kolla på hur mycket de andra verkligen jobbade. Det var även svårt att snabbt få kontakt med andra medlemmar som kunde något bättre än andra. Det i kombination med arbetsbelastning från övriga kurser ledde också fram till en ojämn arbetsbelastning.

Att det inte finns någon beställare utan att vi under arbetets gång skall vara våra egna beställare. Det är därför svårt för någon i gruppen att ta sig an uppdraget som *product owner* och därmed också svårare att utvärdera om den tillagda funktionen verkligen var den som tänktes på från början. Därför missas lite av fördelarna den kontinuerliga leveransen. Det finns liksom ingen beställare som kan utvärdera om det han vill ha har levererats.

Det var svårt att uppskatta hur lång tid alla moment ska ta när man inte är van vid att utveckla för en viss plattform och en sprint blir därför alldeles för stor eller alldeles för liten. Men jag tror att detta komma lösa sig i ett längre projekt eftersom *sprint preview är* till för att ordna sådant. Det var också svårt att bestämma poängen oberoende av vem som skulle ta sig an det då förkunskaperna i vår grupp skiljde sig kraftigt. Min uppfattning är dock att ett självorganiserande lag har lättare att lösa det än ett mer vattenfallsbaserat.

Det kollektiva ägandet av kod och fokusen på *user stories* istället för var i koden och vilka delfunktioner bidrog visserligen till att applikationen snabbt blev funktionell men det bidrog också till dålig och ostrukturerad kod. Detta eftersom medlemmar lade till funktioner som krävde kod i olika delar av applikationen. Delar som medlemmen kanske inte alltid hade så bra koll på. Detta ledde till att nya funktioner lades till vars funktioner redan fanns och även till att funktioner användes fel. Sammantaget blev resultatet att koden blev ostrukturerad och efteråt behövde refaktoreras.

Fokusen på *user stories* ledde också till att ingen riktig arkitektur eller struktur med objektorientering byggdes upp redan från början. Kod lades istället till där det gick snabbast för just den *user story* som jobbades med för tillfället. Detta gjorde koden svår att överblicka senare i projektet och det krävdes därmed allt mer tid för att lägga till något nytt. Struktur fick därför skapas i efterhand med refaktorering.

Teknikens effektivitet

Sammanfattningsvis tyckte jag att metoden var ett väldigt effektivt sätt att jobba på som krävde förhållandevis lite administration. Alla visste hela tiden hur vi låg till och vad som behövde göras. Alla kände också ansvar vilket ökade på effektiviteten. Det kändes helt enkelt som vi fick mycket gjort och det visar sig också i slutprodukten.

Situationer för användning

Jag skulle använda denna teknik i nästan alla typer av mjukvaruprojekt i framtiden. Jag ser främst två huvudtyper av uppdrag/jobb det skulle passa alldeles utmärkt till och en som det skulle passa lite mindre bra till.

Det först är när man arbetar i någon form av konsultroll och ska producera något kunden vill ha. Kunden kanske inte riktigt kan beskriva det den vill ha så bra och kanske saknar kunskaper inom

mjukvaruutveckling. Dessutom kanske programmerarna inte är så väl insatta i kundens teknik och problem. Då fungerar agil utvecklingsmetodik med dess kontinuerliga releaser och utvärderingar alldeles utmärkt. Man kan då minimera jobb som leder till onödiga funktioner och bara lägga tid på det som verkligen eftersöks. Inget jobb görs då i onödan och kunden blir nöjd med resultatet.

Däremot I projekt där en egen produkt utvecklas utan någon riktig beställare skulle jag fundera en extra gång innan jag använde en agil metod. I sådana projekt är det företaget själva som sätter upp kraven på mjukvaran och är då förmodligen mer väldefinierad. Detta eftersom regelbunden utvärdering inte riktigt finns ett behov av här då företaget och utvecklarna själva bestämmer vad som bör finnas. Processen tar här kanske onödigt mycket tid.

I projekt av detta slag med en vision och inga direkta krav på applikationen har det visat sig ovärderligt med en agil metod. Min uppfattning är att appen inte hade varit så enhetlig och funktionsrik utan denna metod. Det visar sig inte minst när jag jämför med den tidigare projektkursen jag har läst där vi inte hade någon tydlig metodik. Det är väldigt stor skillnad på effektivitet och kvalitet på produkten och även i nedlagt tid av varje medlem.

Vad som fungerade bra

Den agila utvecklingsmetodiken var väldigt bra och effektiv. Även versionskontrollen med Git och Github var användbar. Övriga verktyg som användes i kursen var också till stor tillfredsställelse och kommer användas i framtida projekt.

Att vi kommer från olika bakgrund. Det vill säga att vi har olika förkunskaper. Det har gjort att vi kan fokusera på det vi är bra. Olika kunskaper gjorde också att vi fick olika syn på allt vilket bidrog till en bättre produkt. Detta var faktiskt första gången under utbildningen jag arbetade med personer med annan bakgrund och det var mycket positivt trots att programmeringskunskaper ibland saknades. De var dock bättre på struktur och dokumentation vilket vägde upp.

Vad som fungerade dåligt

Att vi kommer från olika bakgrund och läser olika andra kurser. Det medför att vi inte riktigt kan synkronisera jobbet. Det bidrog också till att det var svårt att sätta poäng på olika *user stories* eftersom vi hade olika förkunskaper. Arbetet hade också kunnat göras bättre med alla i laget på samma ställe.

Det enda jag tyckte var mindre bra när det gällde det tekniska hjälpmedlen och verktygen var Android-emulatorn. Den fungerade inte alls med Google Maps API v2 och upplevdes som väldigt långsam. Det i kombination med att alla inte hade en egen Android-telefon gjorde testning stundtals svår. Att Chalmers datorer inte heller hade en uppdaterad version av Android SDK gjorde utvecklingsverktygen otillgängliga. Att fundera på inför kommande instanser vore att installera de nyaste versionerna på datorerna och möjligen tillhandhålla någon billig Android-telefon för utlåning.

Fem personer var möjligen lite för många för ett projekt av denna storlek. Det var som medlem svårt att hålla koll på vad alla jobbade med och det var stundtals några som jobbade på samma sak eller inte hade något att jobba med. Bättre resultat hade kunnat uppnås med färre deltagare. Möjligen är fyra personer max för ett projekt av detta slag.

Förbättringar till nästa projekt

Jag uppfattade avstånden och tiden som det största problemet. Vi var fem individer som hade olika schema vilket gjorde att vi inte kunde vara på samma plats och jobba ihop. Därför skulle jag premiera att alla jobbar samma tider och på samma ställe. Detta för att bli ändå mer effektiva. En annan fördel med att träffas är att ansikte mot ansikte ger den bästa kommunikationen och ingen jobbar på

samma som någon annan. Att alla är på samma ställe leder också till att man kan hjälpa varandra så att var och en alltid jobbar på det han är bäst på. Snabbare lösningar om alla är på plats helt enkelt.

Jag skulle nog strukturerat upp kodningen på ett helt annat sätt. Det blev väldigt rörigt när fokus låg på *user stories* istället på var funktionen skulle finnas. Detta bidrog till att medlemmar implementerade funktioner som krävde funktioner de inte hade riktigt överblick över. Antingen användes funktionerna fel eller så gjordes nya som egentligen gjorde samma sak. Möjligen kan man fokusera mer på vad som behövs för alla funktioner snarare än för funktionen.

En annan sak vilken ska förbättras till nästa gång är att i förväg göra upp en grundläggande arkitektur och struktur för programmet. Vilka delar har ansvar för vad. Som det blev nu var det väldigt rörigt och strukturen fick i många fall fixas i efterhand genom refaktorering. Uppfattningen är att den totala tid med detta angreppssätt var större än om man från början beaktat arkitekturen mer noggrant. En från början uppgjord arkitektur kan sedan allteftersom revideras när mer kunskap kommer till ytan.