Post Mortem Rapport

DAT255 - Software engineering project, HT 2013

Grupp 17

Utvecklingsteamet

Vi var en grupp på 6 personer, fördelade enligt följande:

- 3 personer från Civilingenjör Data, årskurs 3.
- 1 person från Högskoleingenjör Data, årskurs 3.
- 2 personer från Civilingenjör IT, årskurs 1.

Teknisk erfarenhet och kompetens var något spridd, delvis som en konsekvens av ovanstående. I relation till flertalet medstudenter som läser Civilingenjör IT, årskurs 2, hade vi allihop färre kurser i Java-programmering. Ingen i teamet hade heller erfarenhet från vare sig projektkurser i programmering eller att programmera mot Android. En annan konsekvens av spridningen mellan olika utbildningar var att de tidsmässiga snittytorna var relativt begränsade, speciellt för aktiviteter som inkluderade *hela* teamet. Ambitionsnivån hos respektive medlem visade sig tidigt vara genomgående hög. Tidsmässigt var vi allihop beredda att lägga mer än ”Chalmers-normen” på 50/2 timmar per vecka. Viss tvekan fanns kring huruvida den app vi kände starkast för riskerade att ställa för höga krav på teamets tekniska kompetens. Dessutom var vi osäkra på lägsta grad av implementation som krävdes för att motsvara vår vision.

Sett till intresset för att försöka tillämpa och ta till sig agila processer och metoder, speciellt Scrum, var intresset genomgående högt och vi hade alla uppfattningen att det var den kunskapen som var högst prioriterad i den här kursen.

Val av app

Det stod tidigt klart att förslaget att göra en virtuell kortlek var den idé som kändes mest spännande att arbeta med. På grund av sin tekniska natur med kommunikation mellan flera enheter var det dock samtidigt det alternativ som förväntades innebära störst utmaning. Och som därmed även skapade störst risk för att vi inte skulle lyckas implementera något som låg tillräckligt nära vår vision. Då vi till slut tog beslutet att fortsätta med kortleken skedde det baserat på framförallt två faktorer:

- Uppmuntrande ord från handledare.
- En Proof of Concept som indikerade att kommunikationen mellan flera enheter inte behövde bli så komplicerad som vi befarade.

Metoder vi använde oss av under arbetet

Brainstorming

Innan vi hade startat själva projektet hade vi ett par sessioner där vi helt enkelt försökte samla de tankar som fanns och sålla fram sådant som var relevant, dels kring vilken app som skulle väljas, dels kring respektive medlems kompetensprofil och intresseområden.
Vi utsåg tidigt en ställföreträdande projektledare, med uppgift att driva diskussionerna framåt, samt en notarie som skulle sammanfatta och kommunicera vad som diskuterats och beslutats.
Som initial aktivitet, då det framförallt var viktigt att alla fick komma till tals och vädra just sina tankar om hur arbetet skulle fortskrida, passade det utmärkt att ha möten utan fast agenda.

Proof of Concept

Som nämnts var teknikområdet relativt nytt för oss allihop och vi behövde därför försöka öka vårt tekniska självförtroende. Sebastian, en av de mer tekniskt orienterade i gruppen, lyckades med relativt liten insats skapa fungerande kod för att få flera enheter att kommunicera med varann.
I ett senare läge genomförde även vår Scrum Master Fredrik en PoC kring elektroniska verktyg för hantering av Scrum-artefakter.

En PoC kommer främst till sin rätt för isolerade och relativt små problemområden. I gengäld erbjuds ofta snabba resultat med stor betydelse för hur man väljer att fortsätta framåt genom ett område där man saknar kunskap. För att lyckas gäller det att vara noggrann med att identifiera de nyckelområden man vill verifiera. Om man å andra sidan har lyckats med den saken så har man kommit förhållandevis långt på resan mot sitt slutliga projektmål.

När en PoC innefattar utveckling av kod är det viktigt att alla inblandade är införstådda med att det är just ett koncept man fokuserat på, oavsett hur väl resultatet verkar motsvara vad man vill ha i sin slutprodukt. Mycket tid kan tillkomma när det gäller att skapa en slutlig lösning som passar bra in i den givna problemdomänen.

Scrum

Förutsättningar, antaganden och begränsningar

Det stod från början klart att vi ville använda Scrum genomgående i vårt projekt. Utifrån vad som tagits upp under kursens tidiga föreläsningar såg vi helt enkelt ingen anledning att prova något annat.

Jämfört med ett riktigt projekt särskilde sig vår kursuppgift på flera punkter, vissa mer uppenbara än andra. Frågan var hur pass mycket dessa faktorer skulle få betydelse för hur väl vi skulle lyckas samt om vi trodde oss kunna bli tvungna att justera något med utgångspunkt i dessa begränsningar.

Projektets storlek.

Då vårt projekt var väldigt begränsat till sin storlek var det kanske ännu viktigare än annars att undvika att lägga tid på att gå i fel riktning. Dels skulle det finnas väldigt lite återstående tid för korrigeringar, dels var chansen liten att kunskap som uppnåtts i ett sådant sammanhang någonsin skulle kunna göra nytta någon annanstans.

Vidare var det önskvärt att så långt det var möjligt använda standardkomponenter och etablerade lösningar, vilket i åtminstone någon mån begränsade det tekniska rörelseutrymmet och möjligheten att sätta en unik touch på vårt program.

Ur ett investeringsperspektiv begränsades på motsvarande sätt utrymmet för tidiga kostnader, i vårt fall tidsmässiga, eftersom återbetalningstiden i allt väsentligt saknades.

Sprintarnas längd.

De fem halvtidsveckor vi hade att lägga på uppgiften motsvarar grovt räknat en enda sprint i ett normalt projekt. Det är enkelt att konstatera att detta innebar en begränsning av hur många metoder vi skulle hinna med att prova på, särskilt om vi dessutom hade för avsikt att få en rättvis uppfattning om fördelar och nackdelar från var och en. Det skulle helt enkelt inte gå att hinna med att prova alla, inte minst om vi hade för avsikt att i någon mån isolera användningen av respektive.

Prioriteringskonflikter för deltagarna.
Det var uppenbart att vi inte skulle komma i närheten av en situation där vi kunde jobba som ett enhetligt team kring en enda uppgift eftersom vi var distribuerade tidsmässigt. Framförallt förutsåg vi problem med att få till stånd aktiviteter som involverade hela gruppen. Dessutom skulle det även kunna bli svårt att med kort varsel prioritera om och gemensamt sluta upp kring ett eventuellt problemområde.

Att simulera projektets intressenter.
Även om viss input kunde betraktas komma från handledningstillfällena och från själva kursen i sig så skulle vi i teamet alltid vara de som till sist prioriterade saker och ting. I mångt och mycket såg vi detta som en förenkling av vad som normalt kan förväntas vara fallet. Vi skulle exempelvis aldrig behöva förklara eller motivera aktiviteter som vid första anblick kunde ses icke värdeskapande, exempelvis refaktoriseringar eller åtgärder för att tillgodose icke funktionella krav. Vi kunde även vara säkra på att slippa missförstånd i samband med populering av Product Backlog. Vidare kunde många ickefunktionella krav förväntas vara uppenbara för alla inblandade intressenter med minskat behov att dokumentera dem som konsekvens.

Kunskapsnivån inom teamet.
Vi saknade som sagt i princip erfarenhet från Androidutveckling, inte minst från områden specifika för den app vi till slut valde att bygga. Förväntad svårighet att göra tillförlitliga tidsuppskattningar var uppenbar, både när det gällde nedbrytning till tasks men även på övergripande nivå när User Stories skulle läggas in i backlogen. I ett verkligt scenario kan man anta att åtminstone någon expertis brukar finnas tillgänglig om man väl valt att finansiera ett utvecklingsprojekt. I någon mån kunde vi kanske räkna med teknisk handledning men hur det skulle fungera återstod att se. Dessutom var frågan hur långt vi ville dra vår strävan efter att vara ett självförsörjande team.

Det faktum att projektet hade två, delvis konfliktande, leverabler.
Å ena sidan – om vi bara satsade på att få fram en så bra app som möjligt så är det, på grund av projektets natur, möjligen sannolikt att vi skulle lyckas utveckla en app med fler funktioner och av högre kvalitet.

Å andra sidan – om vi lade för stor vikt vid att lära oss om och prova olika metoder skulle vi löpa risk att inte hinna färdigställa ett tillräckligt bra program.

Frågan var också om vi inom teamet hade samma uppfattning av var gränsen mellan dessa bägge leverabler skulle dras.

Val av verktyg
Vi enades om följande:

- Om möjligheten funnits så hade vi, för enkelhets skull och av pedagogiska skäl, föredragit en fysisk Scrum Board. Det var dock enkelt att konstatera att faciliteter för ändamålet saknades. Bland de virtuella alternativen prioriterade vi enkelhet framför avancerade features. Dessutom ville vi att boarden skulle vara web-baserad och färdig att använda utan krav på att installera någonting på egna servrar. Slutligen behövde den ha en licensmodell som matchade våra behov.

- För Product Backlog hade vi egentligen bara behovet av ett verktyg som kunde hantera enkla tabeller. Gick det att hitta något som var integrerat med en intressant Scrum Board var det att föredra.

- Vi valde att initialt inte använda Burndown Charts eftersom vi var väldigt osäkra på vår leveranskapacitet. Ett par veckor in i projektet var denna osäkerhet fortfarande stor. Med

tanke på de korta sprintarna räknade vi även med att det skulle bli svårt att både hinna upptäcka problem och korrigera innan sprinten var slut. Verkningsgraden på tiden det skulle ta att arbeta med detta verktyg bedömdes såpass liten att vi kom att avstå projektet igenom.

- Sprint Backlog behövde uppfylla samma kriterier som Product Backlog.

- Behovet att skapa automatiska rapporter och liknande bedömdes som litet eller obefintligt.

Det slutliga valet föll på Pivotal Tracker, ett verktyg som även nämndes senare under kursen.

Att arbeta med Scrum

Breakdown och tidiga beslut
Det tog en hel del tid och energi att bryta ner vår vision till Epics och vidare till User Stories. Som belöning fick vi allihop god träning i såväl själva arbetssättet som i att åstadkomma tydliga och konkreta formuleringar. Vi ansträngde oss för att inte hasta igenom denna fas av projektet och med facit i hand var det ett klokt val då vi lyckades skapa en stabil bas som vi hade stor nytta av längre fram.

Något annat som vi valde att fokusera på redan från början var begreppet Definition of Done. Det fick helt enkelt ta tid att reflektera över detta. Det visade sig att vi aldrig behövde revidera vår definition men det är inte uppenbart huruvida det kunde funnits fördelar med att låta detta växa fram under gång istället. Som saken föll ut så hade vi åtminstone en sak mindre att fokusera på under projektets gång.

Möten
Vår initiala planering såg ut som följer:

- Sprint Planning måndag förmiddag.
- Leveransmöte+Sprint Review+Sprint Retrospective fredag förmiddag.
- Standup måndag, onsdag och fredag.

Då behovet av fler Standup-möten gjorde sig gällande lyckades vi ofta stoppa in ytterligare ett eller två sådana under resten av veckan. Orsaken att vi valde att göra så var för att vi helt enkelt kände att vi behövde tätare avstämningar med hela teamet av hur vi låg till.

Vår disciplin var överlag god, framförallt under Standup då somliga valde bort schemalagda aktiviteter i andra kurser.

Utöver ovanstående aktiviteter förekom ett antal programmeringssessioner under lite friare former, där den som kunde delta dök upp och så satt vi i grupp, ofta indelade i par, och arbetade. Under dessa tillfällen var det även vanligt att diskussioner av olika slag dök upp och tack vare att vi satt tillsammans var det relativt enkelt att avhandla saker i farten. På grund av nämnda skillnader i schema var det dock vanligast att endast *en del* av gruppen lyckades sitta tillsammans.

Det visade sig så småningom att veckorna blev väldigt komprimerade till vissa dagar. Orsakerna var att relativt få kunde sitta med projektet under torsdagarna, samtidigt som vi hade valt handledning på onsdagar. Dessutom hade vi tidigt bestämt att försöka koncentrera den här kursen till vardagarna, samtidigt som andra åtaganden flyttades till helgerna. Detta för att öka möjligheterna att sitta tillsammans. Konsekvensen av alltihop blev att feedback från handledaren inte alltid kunde omsättas under aktuell sprint utan först veckan därefter.

Den relativt höga mötesdensiteten ledde så småningom till en del justeringar som kommer beskrivas ytterligare inom kort.

Arbetet
Tack vare att vi lagt så pass mycket tid på att bryta ner vår vision var det relativt enkelt att hålla farten uppe under utvecklingsaktiviteterna. Det visade sig så småningom att vi varit lite för defensiva i våra initiala uppskattningar av hur mycket vi kunde hinna med under en sprint och vi korrigerade således detta uppåt för varje sprint som gick.

Som väntat uppstod situationer då vi behövde prioritera om resurser och flytta folk till de tasks som var viktigast. Tyvärr var det, på grund av de ständiga schemakonflikterna, inte alltid enkelt att lösa dessa behov i praktiken. I ett scenario med mer öronmärkta resurser hade läget varit ett annat, nu fick vi inte fullt ut chansen att dra nytta av denna aspekt av ett agilt upplägg.

Kursändring
Efter ungefär halva projekttiden stod det klart att vi behövde prioritera utvecklingsarbete framför arbete med processer och metoder. Fram till den tidpunkten hade det inom teamet rått delade meningar kring hur man skulle relatera sig till kursmålen och projektets leverabler. Efter diskussion med handledaren tog vi utan vidare beslutet att flytta fokus mot utvecklingsarbetet och att acceptera fler utestående frågetecken kring hur man borde göra för att i vissa lägen följa processen fullt ut. Skillnaden i produktivitet var tydlig, samtidigt som vi hade hunnit lära oss såpass mycket om processen att det flöt på väldigt bra ändå.

Det var dock fortfarande så att vi, på grund av det snäva rörelseutrymme projektets tidsram erbjöd, intog en ganska försiktig hållning när det gällde våra arbetssätt. I ett mer förlåtande sammanhang i form av ett längre projekt med längre sprintar hade vi troligen försökt experimentera mer för att se vad som hände.

Utmaningar
Den tydligaste risken med Scrum i en typ av projekt som vårt torde vara att hålla nere mötesdensiteten och att undvika att samlas i onödan. Ett möte på 15 minuter innebar ofta även en omställningstid på ytterligare lika lång tid och i extremfallet att någon tvingades besöka campus utan att ha övriga ärenden dit den dagen.

Samtidigt hade vi ett stort behov av att träffas under åtminstone några former, mycket för att vi ofta fick testa oss fram med tekniken eller för att oförutsedda frågeställningar dök upp. I vårt fall löste det sig ganska bra genom att många i gruppen ofta samlades spontant för att jobba med utveckling med följd att frågor som i andra sammanhang skulle behövt avhandlas under ett regelrätt möte istället kunde klaras av där och då. I någon mening är det väl även ett arbetssätt som pekas ut av processen.

Det var dock fortfarande så att mötesmängden, framförallt till antalet, var väldigt stor i förhållande till projektets övriga förutsättningar och i förhållande till den uppenbara avkastningen. Dock hade vi tidigt bestämt att följa Scrum så långt som möjligt, varför vi inte såg anledning att göra annorlunda.

eXtreme Programming

Många av dessa practices var intressanta men vi var tidigt överens om att försöka prioritera de som kändes viktigast och sedan antingen hålla oss till dem under hela projektet eller att försöka variera användningen av bara ett fåtal för att enklare kunna bedöma nytta och fördelar. Ett par av dem var i någon mån anammade av samtliga gruppmedlemmar redan före starten av denna kurs, framförallt de som gäller *Shared Understanding* och *Testing*. Andra låg så pass nära Scrum-

processen att de följde med i det sammanhanget utan att behöva beaktas separat, exempelvis *Planning Game* och *Small Releases*.

Practices vi övervägde men avfärdade

Programmer Welfare
Rimmar till att börja med illa med Chalmers 50-timmarsveckor. Dessutom var vår ambitionsnivå så pass hög att det var tvunget att finnas utrymme att lägga den tid som vi tyckte behövdes, utan att samtidigt behöva riskera andra kurser. Det gällde även ett projekt med garanterat kort livslängd som dels var garanterat att ta slut, dels knappast kunde förväntas erbjuda lugnare perioder där motsvarande lägre tid kunde spenderats.

Continuous integration
Då vi redan hade väldigt täta releaser separerade av, förhållandevis, få utvecklingstimmar såg vi inte att vinsten med denna metod skulle kunna räknas hem. Däremot ansträngde vi oss för att hålla vår Dev-branch väl uppdaterad i bägge riktningar med följd att mergningar ner till Master alltid gick snabbt och smärtfritt. Med lite vilja skulle man kunna säga att vi, åtminstone delvis, tillämpade Continuous Integration i Dev. I ett större projekt är fördelarna uppenbara men som saker såg ut för oss så fick det bli en sådan kompromiss utan denna metod för att inte hindra oss i onödan.

Practices vi provade men som kanske inte gav önskad avkastning

Test Driven Development
Då denna metod verkar relativt utbredd i riktiga projekt hade det varit intressant att prova på det ordentligt men vi konstaterade tidigt att arbetssättet innebar en extra belastning som vi inte ville lägga på oss själva. Enligt oss har TDD en relativt hög tröskel som dels innebär att en initial investering måste betala av sig, dels ökar osäkerheten kring huruvida man skall hinna med utveckling av exempelvis features där det råder stor tveksamhet kring vilken lösning man kommer landa vid. Ironiskt nog finns även risk att tiden att testa sig fram *kring en viss lösning* blir begränsad.

Automatiska GUI-tester
Initialt var ambitionen att försöka använda detta så mycket som möjligt, som komplement till övriga tester. Dock visade det sig efter ett tag att de verktyg som fanns att tillgå var svåra att få att fungera med oförutsägbara resultat som följd. Allra störst problem hade vi när vi kombinerade GUI-testerna med Android-emulator.

Efter att ha lagt ner väldigt mycket tid på att få dessa tester att fungera skiftade vi senare fokus mot sådant i projektet som vi enklare kunde påverka. Mängden manuella GUI-testfall ökade i motsvarande mängd.

De practices vi provade som hade störst positiv effekt

Pair Programming
Utöver att vara ett roligt sätt att arbeta var detta även ett bra sätt att får två personer att jobba effektivt kring samma problem då den ena av dem kunde leta information och lösningar medan den andra kunde skriva kod och provköra kontinuerligt. En annan positiv aspekt var att två personer kunde använda samma telefon med allt vad det innebar.

Den andra sidan av myntet innebar emellertid att det av tidigare nämnda skäl var bitvis knöligt att få till stånd denna typ av aktiviteter.

Den fas i projektet då vi tyckte denna metod hade störst berättigande var i början, då det fortfarande fanns relativt få och centrala tasks. Dels för att det inte fanns nog avgränsade uppgifter åt alla, dels för att det gällde centrala delar av applikationen där kunskapsspridningen var som viktigast.

Vi upplevde emellertid att detta arbetssätt ställde höga krav på gemensam närvaro och tog halvvägs in i projektet ett medvetet beslutet att försöka släppa kravet på att följa det. En märkbart större schemamässig frihet för den enskilde kunde omedelbart noteras, samtidigt som det möjligen även innebar en ökning av produktionstakten. Då hade vi även kommit så pass långt i projektet att varje enskild person kunde arbeta med och checka in isolerade delar av programmet utan att löpa nämnvärd risk att störa andra.

Small Releases
Överlappar ju Scrum och vad som är brukligt där, så valet att försöka applicera detta var ju i det närmaste givet på förhand.

Något som stack ut tydligt när det gällde att jobba i små steg var att osäkerhet blev lättare att hantera. Dels kunde vi bolla våra funderingar med handledarna, dels hade vi hela tiden en ganska tydlig bild av hur vi låg till i relation till projektets övergripande tidsplan och den vision vi försökte förverkliga. Problem identifierades tidigt och kunde ges rätt prioritet utan att orsaka panik. Negativa effekter av vår bristande erfarenhet mildrades påtagligt av att vi hela tiden automatiskt hölls uppdaterade med hur vi låg till.

För att säkerställa att vi alltid hade refaktorisering i åtanke lade vi från början in detta som en punkt i vår Definition of Done. Risken med inkrementell utveckling är annars att man prioriterar ner att snygga till fungerande kod eftersom behovet att öka kvaliteten kanske inte är lika uppenbar som behovet att få klart ytterligare en ny feature.

Planning Game
Ytterligare en practice som i mångt och mycket överlappar vad som kännetecknar Scrum. Här hade vi fördel av att de enda egentliga intressenterna var teamet självt och att det var enkelt att slå fast värdet av en viss feature eller User Story. Som nämnts var det svårt att göra rättvisa tidsuppskattningar men oftast fick businessvärdet väga tyngst för oss och överlag fungerade det bra, eftersom osäkerheten kring arbetsinsats sträckte sig både uppåt och neråt med viss utjämning som följd.

All code must have Unit Tests+ all code must pass all Unit Tests before it can be released
Här hade vi som sagt ganska stora problem med verktygen men bortsett från det så lade vi stort fokus på att etablera ett arbetssätt där vi tog testning och testfall på allvar. Initialt tog det ganska mycket tid att skriva underlagen men efter hand gick det smidigare och kom till slut att bli en helt naturlig del av processen.

Funderingar kring nackdelar och risker med XP

Det tar tid att få grepp om vad som fungerar under specifika omständigheter.
Då XP verkar relativt situationsberoende är det svårt att enbart utifrån ett teoretiskt underlag snabbt skaffa sig en bild av vilka practices som fungerar i vilka sammanhang. Det verkar därmed finnas två sätt att istället skaffa sig sådan kunskap - antingen att prova sig fram eller att involvera en erfaren expert. Även om vi hade stöd från handledare som i någon mån kunde utgöra expert så var vi till största delen utlämnade till det återstående alternativet men här var

projektets natur ett reellt hinder. Vi hade föredragit ett längre projekt att prova i eller mer omfattande engagemang från någon med praktisk erfarenhet, helst såklart en kombination av bägge.

Oprecisa estimeringar och "Scope Creep"
Spelar rimligen in mest i sammanhang då projektmedlemmarna är oerfarna. Å andra sidan så hade väl ett upplägg á la vattenfall bara gömt problemet och skjutit upp det eftersom den tid som hade krävts för att uppnå säkerhet i ett sådant sammanhang knappast varit mindre, bara fördelad annorlunda och dessutom med förväntat sämre faktiskt utfall som resultat.

Avsaknad av dokumentation
Med fullt fokus på snabba leveranser är det lätt att helt tappa fokus på att dokumentera och kommunicera sådant som är viktigt eller på annat sätt kan motiveras ur ett helhetsperspektiv. Konsekvensen kan bli öar av kunskap inom teamet och ibland att felaktig information lämnas till diverse intressenter utanför utvecklingsteamet. Därmed tar det exempelvis längre tid att upptäcka tveksamma designval, inte bara för teammedlemmarna utan även i sammanhang där utomstående parter av någon anledning granskar lösningen. Listan av konsekvenser slutar inte där men i slutänden handlar det om att man riskerar dels programvarans kvalitet, dels projektets budget och tidplan.

I vårt specifika projekt skulle konsekvenserna av denna typ av försumlighet ha blivit ganska små, mest beroende av att det program som hann byggas blev såpass litet.

Att jobba i grupp

Förutsättningar
Sett till olikheterna inom gruppen så var spridningen relativt liten. Fördelen var att det ofta gick enkelt att ta beslut och komma överens, nackdelen såklart att det fanns viss risk för ett begränsat synsätt, jämfört med ett sammanhang med mer varierande åsikter. I ett så pass kort projekt övervägde dock fördelarna med att snabbt kunna komma framåt. Helt befriade från fördjupade diskussioner var vi heller inte.

Uppstart
Då alla i princip delade samma ambitionsnivå gick det fort att komma överens om var ansträngningsnivån skulle ligga. Dessutom hade alla relativt lätt att ta upp saker för diskussion och det gick överlag relativt snabbt att sätta de första pinnarna i marken.
Det faktum att vi tidigt försökte sätta en struktur för kommunikation och möten gjorde att vi slapp dra samma diskussioner flera gånger utan istället kunde ägna oss åt att komma framåt i andra frågor.

Under gång
Vartefter arbetet fortgick blev det alltmer uppenbart att det fanns lite delade meningar kring kursmålen och projektets leverabler. Förvirringen ökades även av att vi alla saknade tidigare erfarenhet från denna typ av projekt. Här hade vi stor nytta av att vi kunde bolla våra funderingar med handledare, något som sedermera resulterade i nämnda fokusskifte från djupare förståelse av processer och metoder och mot högre leverans av features och funktionalitet.
Vi kom aldrig till lägen med regelrätta konflikter, kanske för att projektet var så pass litet till sin omfattning. Toleransen var relativt hög när det gällde att låta någon ta upp en frågeställning. Vid

djupare argumentering var vi överens om att ta upp saken med handledare och oavkortat försöka följa de råd vi fick därifrån.

Andra faktorer med betydelse för vårt projekt

Tid nedlagd på att utvärdera för projektet vitala verktyg

Vi hade framförallt problem med två verktyg, Android-simulatorn och ramverken för automatiska GUI-tester. Tidigt i projektet togs beslutet att vi skulle lägga oss på en viss Target API och vi utgick då från att Android-emulatorn och ramverket för automatiserade GUI-tester skulle fungera någorlunda tillfredsställande. När så inte visade sig vara fallet hade vi redan hunnit för långt in i arbetet för att före deadline kunna räkna hem en ändring av API-nivån. Hade vi fått gå tillbaks och göra annorlunda hade vi försökt lägga lite tid på en PoC även för vissa av våra utvecklingsverktyg. Straffet blev nu att väldigt mycket tid försvann i onödan i ett läge när det var väldigt lite vi kunde göra åt saken. För dem som drabbade hårdast av problemen med simulatorn föll det sig naturligt att i större utsträckning fortsätta med par-programmering.

Testning och verifiering tar tid

Även om vi från början hade relativt stor medvetenhet kring vad som behövde göras överraskades vi av hur mycket tid det faktiskt går åt till att utföra dessa sysslor till belåtenhet, även obeaktat den tid som försvann då vi hade problem med de automatiska testerna. Även om man är hjälpt av automatiska tester är det fortfarande så att lejonparten av arbetet går åt till att fundera på *vad* och *hur* man vill testa, inte att skriva koden för ändamålet.

Icke funktionella krav är inte underförstådda

I vårt specifika projekt var detta extra svårt att hålla i minnet eftersom det för någon som sysslat mycket med programmering och är i allmänhet tekniskt involverad kan vara svårt att enkelt identifiera de kvalitetsfaktorer som kanske inte är självklara för någon på exempelvis beställar- eller användarsidan. Här gällde det att främst identifiera och tydliggöra de ickefunktionella krav som var *viktigast*, snarare än att försöka få med allt.

Det tar tid att sätta en god arkitektur

Vi fastnade tidigt i projektet på hur vi ville att programmets övergripande struktur skulle se ut. Mycket tid lades på diskussioner och argumentation utan att det kändes som om vi kom framåt. Men så gick plötsligt proppen ur och vi upplevde ett moment of clarity där allt föll på plats. Här kan man fundera på konsekvenserna av ifall vi hade gett upp tidigare och valt antingen en arkitekturell kompromiss som vi inte alla kände för eller, i värsta fall, tvingats till stora avvikelser från vår vision. När vi ser tillbaks så är vi glada att vi lät denna del av arbetet ta den tid som behövdes. Vi såg i slutändan att detta noggranna förarbete gav en tids- och kvalitetsvinst som överträffade den initiala investeringen.

Det är inspirerande att se programmet växa fram

Den tidiga kontakten med en faktisk produkt bidrar till att hålla intresset levande hos såväl utvecklingsteamet som, i förekommande fall, beställare och sponsorer. Dels var det positivt för moralen i utvecklarteamet, dels blev handledartillfällena mer givande när det fanns konkret

funktionalitet att diskutera. Jämfört med vattenfallsutveckling så slipper man långa tider av förberedelser innan man faktiskt ser något konkret ta form.

Nedlagd tid

Första veckan gick åt till att bilda en grupp, så då kunde vi inte börja jobba med projektet. Totalt spenderade vi ca 25 timmar per person de 2 första veckorna på att bestämma vilken app vi skulle göra och att skriva visionen för denna.
Under de följande veckorna lade vi i genomsnitt ca 25 timmar per vecka och person på kursen:

- Av dessa lades ca 3-4 timmar på den teoretiska delen, alltså föreläsningar och annat inhämtande av information, samt på att föra logg över arbetets gång, använda metoder, tagna beslut osv.

- 5 timmar lades på hantering av artefakter i samband med respektive release, alltså sammanställande av tester och dokument, mergning av kod, Definition of Done, osv.

- 5 timmar gick åt till diverse möten som Daily Standups, Sprint planning, Sprint Review, Sprint Retrospective.

- Resterande dryga 10 timmar lades på utveckling av features och tester samt refaktorisering av befintlig kod.

Fördelningen av tid var individuell där några arbetade mer med programmering medan andra lade mer fokus på övriga uppgifter. Problemen med verktygen fick också i någon mån styra hur arbetet delades upp inom teamet. De som hade egna telefoner kom till exempel att naturligt bli mer inblandade i sådant som rörde testning och felsökning.

Här skall man ha i åtanke att samtliga medlemmar var involverade i arbetet kring Product Backlog. Ett alternativt tillvägagångssätt kunde annars ha varit att Product Owner ensam skötte hela Product Backlog och dess underlag samt gjorde prioriteringar men vi valde medvetet att involvera hela teamet i denna process.

Övriga reflektioner

Överlag fungerade arbetet i projektet väldigt bra, trots att förutsättningarna i kursen troligen ligger en bit ifrån vad som torde vara brukligt i riktiga utvecklingsprojekt. Lärdomen blir att agila metoder som namnet antyder verkar väldigt användbara även under icke-optimala förutsättningar. De begränsande faktorer vi upplevde under kursens gång hade förmodligen haft *ännu* större negativ effekt om vi arbetat på ett sätt som låg närmare vattenfallsmodellen, undantaget koncentrationen av möten.

Den förenkling av verkligheten det innebär att sakna påverkan från yttre stakeholders hade påtaglig betydelse för möjligheten att hålla farten uppe i projektet. Givet den korta tid vi hade på oss var det förmodligen nödvändigt med ett sådant upplägg men samtidigt saknades en viktig del av den helhet man befinner sig i när något skall utvecklas för riktiga pengar och levereras till riktiga beställare med varierande erfarenhet och teknisk mognad. Förvisso prioriteras i sådana lägen fortfarande arbetet i samband med sprintplaneringarna men de gånger beställaren har dålig kunskap om hur man skall arbeta med Scrum tillkommer en osäkerhetsdimension som vi här slapp uppleva.