Post Mortem Report

1. Processer och rutiner under arbetets gång

Mjukvaruutveckling av större system är i grunden är relativt komplext. Detta i och med att det lätt kan bli ett flertal komponenter och tekniker som behöver samverka med varandra, det är det därför även viktigt med struktur och regler för utvecklingsprocessen. Genom att använda sig av olika processer och rutiner inom utvecklingsteamet kan man på så vis ge sken för projektmedlemmarna att processen är mindre komplex, och då även förenkla arbetet med mjukvaran.

Hur bör man då gå till väga för att på bästa sätt kunna hantera projektprocessen? Detta är givetvis beroende på projektets omfattning och komplexitet, och i just vårt fall har projektet varit tvunget att vara anpassningsbart. Detta tack vare att samtliga projektmedlemmar har varit nya inom hur man hanterar teknologierna som har använts i projektet, men även hur det är att, på ett effektivt sätt, jobba i ett projektteam. Då projektprocessen har haft kravet på sig att vara anpassningsbart är har det tack vare detta varit lämpligt att tillämpa en empirisk processtyrning (agil processutveckling). I förhållande till en fördefinierad processtyrning (s.k. ”vattenfall”) lämnar det agila arbetssättet mer utrymme för flexibilitet under hela projektets levnadscykel, och kräver inte heller 1en allt för tung förundersökning innan man ger sig på skapandeprocessen. Detta har i vårt fall varit en bra metodik då det har varit begränsat med tid för att utföra projektet, det har därför varit vart viktigt för oss att ge oss på utvecklandefasen så fort som möjligt utifrån en provisorisk projektvision som sedan har justerats under hela projektprocessen.

Som förväntat av oss har vi under projektets gång använt oss av ett agilt arbetssätt med huvudinspiration hämtad från SCRUM-metodiken. Trots att ingen av de inblandade i projektet hade någon större kunskap sedan tidigare kring hur man jobbar agilt har det ändå fungerat bra, och det går definitivt att finna flertalet fördelar med att jobba agilt kontra att jobba med en klassisk vattenfallsmodell.

1.1. SCRUM med extreme programming (XP)

Även fast både SCRUM och XP är agila arbetsmetoder har vi funnit inspiration till en bra arbetsmetodik genom att kombinera olika element från de både metoderna. SCRUM är förvisso en förhållandevis fri arbetsform, men bidrar ändå till en vettig grundstruktur för dokumentation och arbetsstruktur. Genom att använda sig av en *product backlog* där man via *user stories* beskriver vad man som användare i slutet kommer vilja ha ut av produkten är det på så vis enkelt för alla som är inblandade i projektet att se vad som finns kvar att göra. Man kan på så vis se hur arbetet ligger till och vad som behövs göras.

Genom att dela upp dessa *epics,* som man beskriver i sin product backlog, och sedan dela in dessa i flera olika deluppgifter kan man på så vis för varje vecka enkelt bestämma sig för vilka features man vill fokusera sig på inför veckan. Det är på så vis lätt att kunna skapa en struktur över vad man ska göra under arbetsperioden, och dela upp arbetet i flertalet *sprints* som man kallar det inom SCRUM. För oss har en sprint inte stäckt sig längre än under en vecka i och med att vi har haft som mål att ha en release per vecka, vilket även är ett kursspecifikt krav.

Något vi har anammat från XP är bland annat att hela utvecklingsteamet jobbar tillsammans med alla uppgifter som behövs ta hand om istället för att bara sköta det som man själv har blivit tilldelad för veckan. Detta har då även bidragit till att vi alla har suttit samlade tillsammans för att jobba med projektet. Även *pair programming* är en arbetsteknik som vi har hämtat från XP.

Att hålla kontinuerliga möten varje dag (*daily scrum*) är även en rutin som vi har haft under projektets gång. Detta har uppfyllts genom att varje projektmedlem i början av ett arbetspass har berättat för de andra om vad de har gjort sedan senaste arbetspasset och vad de ämnar göra tills nästa arbetstillfälle. Om det har funnits något hinder som har hindrat arbetet har även detta tagits upp samt hur man bör jobba för att lösa problemet.

1.1.1. Fördelar kontra nackdelar med SCRUM/XP

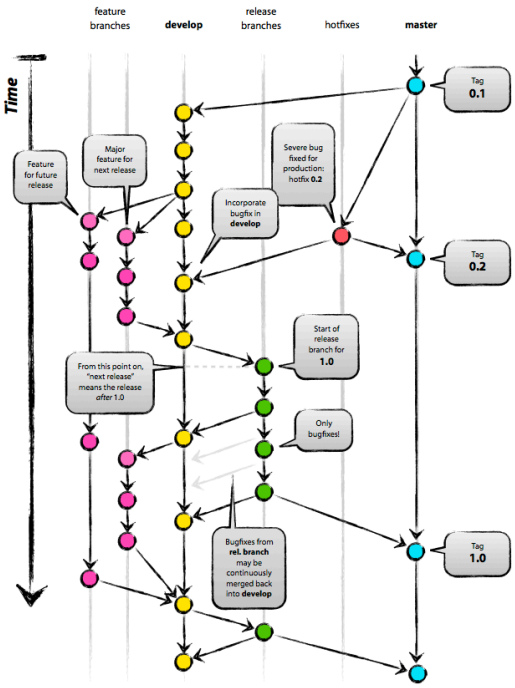
Det vi under denna projektperioden bland annat har reagerat över är hur pass effektivt det är med pair programming. Att låta två personer jobba med samma del av projektet har för oss varit väldigt tidseffektivt då två hjärnor oftast jobbar snabbare än vad en gör. Då problem har uppkommit har dessa oftast kunnat lösas mer smärtfritt än ifall man som ensam programmerare skulle ha attackerat problemet. Då vi även under våra dagliga scrum-möten har förklarat för varandra vilka slags problem som vi har stött på, har vi gjort som så att ett par utvecklare har blivit tilldelade uppgiften att lösa problemet efter mötet för att sedan kunna fortsätta med sprintmålen.

Även fast XP-metodikens tanke om att sitta och jobba mycket tillsammans kan vara effektivt då alla projektmedlemmar enkelt kan konversera idéer och enklare fatta större beslut tillsammans, har vi sett att det inte alltid har varit en bra arbetsmetodik. Då projektmedlemmarna i vår grupp har haft olika scheman och olika dagsrutiner har det tidvis varit problematiskt att hitta en tid som passar samtliga inblandade. Detta har vissa veckor lett till inaktivitet vissa dagar, medan andra dagar har blivit mer hektiska. Det har varit möjligt för varje enskild programmerare att jobba själv hemifrån, men många gånger har man velat checka av med övriga gruppen inför större beslut, vilket har betytt att utvecklingen av just denna delen har skjutits upp tills ett gruppmöte.

Sprintplanerna som man jobbar mycket med inom just SCRUM är en process som vi har funnit mycket användbar för oss. Då varje sprint har varit en vecka lång har dessa sprintplaneringar blivit som veckoplaneringar istället där vi har kunnat bestämma vad vi ska hinna med varje läsvecka. Genom att ta nya user stories från vår product backlog och dela upp dessa i features har vi enkelt kunna ha kontroll över vilka features som har behövts ta tag i. Det har alltså varit ett smidigt sätt att se vad det är vi har kvar att jobba med och hur vi ska dela upp arbetet vecka för vecka. Något som egentligen borde ha understrukits mer av oss under arbetsprocessen skulle ha varit när en feature faktiskt är “klar” så att man kan släppa den och gå vidare på nästa del av projektet. Vissa gånger har en feature utvecklats till fler och fler features under tidens gång, något som inte har tagits i bejakande vid veckoplaneringen. Detta har alltså bidragit att vi i slutet av veckan fortfarande har haft features som inte varit helfärdiga från sprinten. Det har dock inte varit något kritiskt problem för oss då vi projektmedlemmar under hela arbetsprocessen har diskuterat kring vad som behövs göras, som nämnt tidigare.

1.2. Git Flow

För att alla i utvecklingsteamet enkelt ska kunna följa arbetsprocessen och jobba med olika delar av projektet samtidigt bestämdes det tidigt att vi skulle följa arbetsmodellen *Git Flow* för versionshanteringen på vår Git. Den allmänna uppfattningen om arbetsmetoden är mycket positiv då det har gjort projektet lätthanterligt och lättöverskådligt.



Genom enbart använda master-branchen för större programreleaser har det på så vis varit enkelt för oss att se hur projektet har utvecklats vecka för vecka.

Allt arbete har utgått från developbranchen, där gruppmedlemmarna sedan själva har grenat ut till olika feature-brancher. Detta är i stora drag vad vi har utnyttjat av Git Flow i just det här projektet, dvs develop-branchen, masterbranchen och feature-branches. I vårt fall har det ansetts redundant att blanda in release-brancher då sprinterna har legat så nära inpå varandra. Det har därför räckt för oss att sammanställa alla färdiga feature-brancher inför varje ny veckorelease och sedan merge:a in develop-branchen på master.

Något med just denna Git-modellen som på ett plan har varit negativt för oss har varit att det har gett en missvisande bild om hur långt vi har kommit i projektet varje vecka. För detta finns det dock utrymme för diskussion, projektet har givetvis varit lika långt framkommen även fast alla features inte har funnits på master-branchen. Dock har det varit så för vår del att viss funktionalitet som har funnits med i veckans sprint inte har kunnat bli sammanfogad till master-branchen då funktionaliteten i fråga inte har varit färdigimplementerad. Detta är dock snarare något som vi i projektgruppen skulle kunnat ha gjort bättre, nämligen att minska antalet features per vecka eller lägga upp sprintarna något bättre för att på så vis kunna pusha upp all funktionalitet vi har jobbat med under veckan på master-branchen i slutet av sprinten.

1.2. Praktiska användningsområden av processverktygen

1.2.1 När bör de användas?

Scrum och XP som vi har använt oss av ter sig väldigt bra i medelstora grupper där det finns så pass många deltagare att det är svårt att hålla i huvudet vad alla håller på med. I vårt fall har projektgruppen förvisso varit någorlunda stor i det avseendet, men i och med att vi i gruppen umgås så mycket utöver studierna har det inte varit några problem att hålla koll på vad alla gör oavsett.

För oss har det fungerat bra med detta agila arbetssätt då alla projektdeltagarna har varit någorlunda nya kring hur teknologierna som har använts fungerar. Genom att man tillåts att vara flexibel och kunna ändra projekttaktik under hela processen så har det därför fungerat bra. Även fast man har gett sig in i ett problem med en viss grundidé så har inte detta hindrat oss från att senare omformulera lösningsförslaget då vi har blivit mer insatta i problematiken och tekniken.

1.2.2 Vad hade hänt om man hade använt projektverktygen på enbart en liten del av projektet och inte på projektet som helhet?

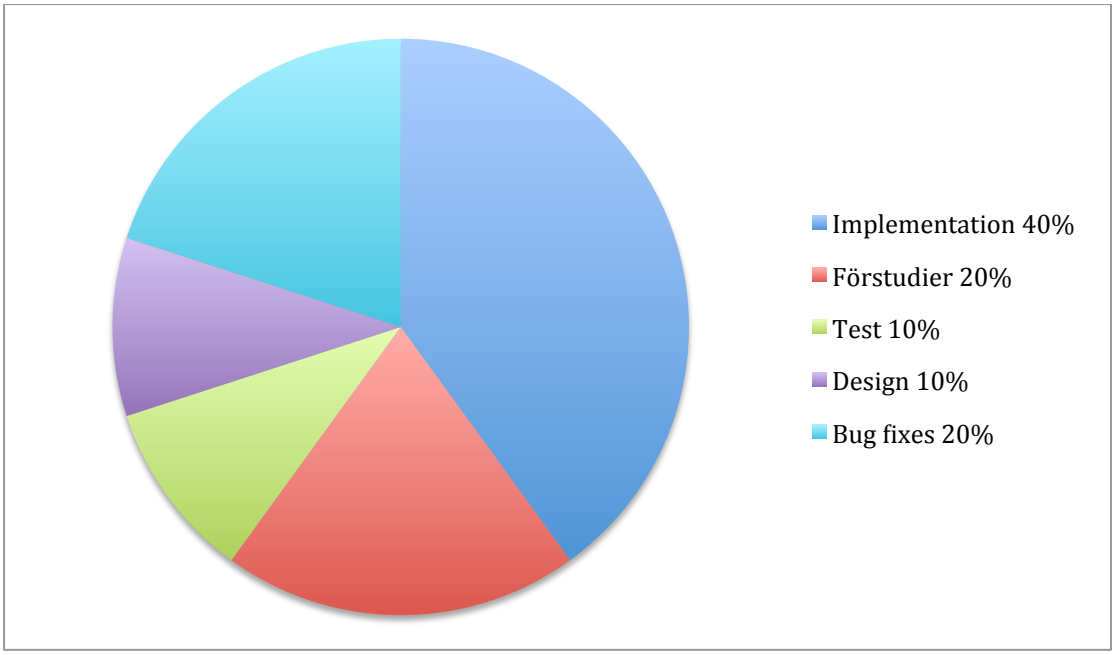
Kommunikationen hade antagligen lidit väldigt mycket och koden hade blivit väldigt individualiserad i den delen av projektet som Scrum och XP inte hade använts. Detta baserat på att man inte hade gått igenom vad som gjorts föregående vecka, eftersom vi använde oss av vecko-sprints, och att man hade arbetat ensam med koden istället för att programmera i par. Det hade betytt att endast en person hade haft insyn i koden som personen ensam har skrivit till skillnad till om man har varit två stycken.

Scrum och XP gör det mycket enklare att ha en överskådlig syn på hur projektet håller på att utvecklas då man, i stort sett, hela tiden vet om vad de andra projektmedlemmarna har arbetat, eller arbetar, med.

2. Tidsåtgång

För att göra en representation av hur mycket tid vi har lagt på varje delmoment i projektprocessen kommer vi använda oss av ett poängsystem där 100 poäng är den totala tiden vi har lagt ner.

Tidsåtgång per moment



Arbetade timmar per person:

Adam Jilsén:	~20 h/w
Tobias Tikka:	~20 h/w
Sebastian Bellevik:	~20 h/w
Marcus Olsson:	~20 h/w
Tomas Hasselquist:	~19.5 h/w
Philip Ekman:	~20 h/w

2. Icke-processpecifika val

LibGXD

Vi valde att använda oss av ramverket LibGDX som grund i spelutvecklandet. LibGDX är ett färdigt ramverk för utveckling av spel till flera plattformar. Möjligheten att kunna köra spelet på flera plattformar gav oss en stor fördel då vi kunde testköra spelet direkt på datorn. Detta är en betydligt smidigare process än att behöva kompilera och köra spelet på telefonen. Ytterligare en anledning till att vi valde LibGDX är att det är ett välkänt ramverk med rikligt med dokumentation och ligger till grund för ett flertal större speltitlar.

LibGDX innehåller ett färdigt system för utritning av grafik och uppspelning av ljud. Det tillhandahåller även en slinga för all spellogik.

Det rent grafiska består av två lager; det första lagret är representationen av spelvärlden, det andra är användargränssnittet. Denna separation mellan lagren gör det enkelt att utveckla de olika delarna var för sig.

Box2D

LibGDX har även fysiksimulatorn Box2D inbyggd. Vi använder den för att representera den logiska spelvärlden och dess olika objekt. Det är sedan denna representation av världen som vi låter presenteras grafiskt för användaren med hjälp av grafikrenderaren OpenGL.

Box2D låter oss applicera verklighetstrogen fysik i spelet genom att ge objekt olika egenskaper, så som massa och friktion. Fysiksimulatorn tillhandahåller även ett färdigt system för kollisionshantering.

Androidemulator

När man utvecklar applikationer till Android finns det två sätt att testa och köra applikationen; via en emulator direkt i datorn eller på en faktisk telefon med Android.

Då denna emulator är känd för att vara extremt dåligt optimerad och långsam valde vi att under hela utvecklingsfasen istället testköra spelet på en telefon, i de fall vi inte ville testköra spelet på en dator. Dessa fall innefattar kompabilitets- och prestandatestning.

Tiled

Tiled är ett verktyg för att skapa spelkartor i vilka världarna är uppdelade i rutnät. Vi valde att använda oss av detta verktyg då LibGDX har inbyggt stöd för den typen av spelkartor. Det är även ett väldigt enkelt och intuitivt sätt att skapa spelkartor.

Spelkartor uppbyggda på detta sätt består av flera lager, vilket gör det enkelt att separera golv från tak och likande saker. Vi använder även denna funktion för att utlösa olika händelser på specifika platser på spelkartan.