2017

Mjukvaruutveckling med Scrum som arbetsmetod

KTH, Projekt IT

Benedith Mulongo
KTH

innehållsförteckning

Innehåll

Projektarbete med metoden Scrum

Inledning

Allmän bakgrund

Under två parallella kurser, Projekt IT och Mjukvaruutveckling, har vi bekantat oss mer generellt med mjukvaruutveckling och i synnerhet med hur ett IT-projekt går till med metoden Scrum. Vi har under fyra veckor använt metoden Scrum i en grupp av åtta personer för att utveckla en mjukvaruprodukt som i vårt fall var ett spel. Spelet var utvecklat med hjälp och stöd av spelmotorn Unity samt programmeringsspråket C#.

Scrum är en agil metod som i nuläget används i stora tekniska och icke-tekniska mjukvaruprojekt i många väletablerade företag runt om i världen. Den är mindre byråkratisk, i det avseende att man dokumentera väldigt lite, man kan lätt anpassa projekt, dessutom kan man snabbt få återkoppling från olika intressenter. Direkt och kontinuerlig återkoppling med olika intressenter underlättas av det faktumet att Scrum bygger på korta interaktioner som kallas Sprint, som sträcker sig över hela projektet. Vid varje sådana sprint, kan olika intressenter tycka till, vilket gör hanteringen av eventuella missförhållande och missförståelsen mellan olika inblandade enklare att hantera. Utöver det, underlättas åtgärden av eventuella fel.

Jag ska i den här rapporten beskriva hur Scrum har underlättat vårt arbete, vilka metoder vi har använt, vilka teorier som vi har utgått från, hur relaterat vårt projektarbete är till tidigare arbete med samma metod. Lite längre fram, kommer det att finnas en beskrivning av vad tiden har gått åt på samt vilka förbättringar som skulle kunna införlivas för effektivisering av arbetet.

Specifik bakgrund

Den agila metoden som Scrum är en del av är karakteriserat av kontinuerlig feedback och ändringar under korta iterationer under hela perioden projektet varar. Empiriska studier av utveckling av mjukvaror med stöd av agila utvecklingsmetoder har inte varit så djupgående[1]. Det krävs vidare studier som visar på hur ett mjukvaruprojekt utvecklas och fortlöper med agila metoder generellt och med Scrum i synnerhet.

Problem

Det stora problemet vi ska här behandla beror mest på bristen på fakta om hur varje enskild medarbetare i en agil projektgrupp arbetar. Vi ska försöka, med utgångspunkt från vårt eget projekt, analysera hur distribution av arbetet går till för att klara leverans av mjukvarusystem samt hur disponering av tiden till olika arbetsuppgifter under projektet kan göras.

Syfte och uppsatsens struktur

I den här rapporten undersöker vi om distribution av arbetet i ett projekt med Scrum som arbetsmetod. Vi ska vidare undersöka hur man disponera tiden åt varje uppgift samt hur en projektgrupp kan bättre använda sig av sina medlemmar för att effektivisera arbetet med utgångspunkt från vårt eget projekt. Först kommer jag att undersöka de metoderna vi har använt oss av, sedan kommer jag att ge en allmän beskrivning av den teoretiska bakgrunden som agila metoder och Scrum bygger på. Efter det, kommer en diskussion om vårt arbetssätt med agila metoder och om hur relaterat vårt projekt är till tidigare arbete med Scrum. Resultat och data vi har samlat kommer att redovisas och senare följer en övergripande analys med förslag på eventuella förbättringar.

Metod

Inledning

För att göra ett bra projekt som möjligt har vi använt oss av olika metoder, dels för att mäta våra framsteg under projektets gång dels för att ha ett stöd eller snarare en mall att utgå från under förarbetet, huvudsakliga arbetet samt efterarbetet. De metoderna som vi följt har varit till mycket hjälp för att slutföra vårt projekt och nå vårt slutgiltiga mål, som i vårt fall var att producera och publicera ett välfungerande spel.

Använda metoder

Vi har i vårt projekt använt följande allmänna metoder:

- Agila metoder och Scrum
- SGDF
- Essential Kernel

Agila metoder och Scrum har vi använt under hela projektet, innan och efter. SDGF, sefl-governance Developpement Framework har använts för planeringen av arbetet samt distribution av tiden åt olika arbetsuppgifter för varje utvecklare. Det har använts flitigt, speciellt under förarbete när vi planera in produktbacklog och prioritera olika tasks. Essential Kernel har vi använt efter varje sprint[1] för att exakt veta status på vårt projekt samt evaluera tiden som bör läggas på varje uppgift för att nå vårt mål. En heltäckande beskrivning av varje använd metod ges nedan.

a. Agila metoder och Scrum

Agila metoder är relativt nya introducerade metoder inom mjukvaruutveckling, de introducerades 2001 [2] i ”det agila manifestet ” [3]. De agila metoderna skiljer sig från mer traditionella metoder såsom vattenfallsmetod, i det avseende att de bygger främst och först på fyra grundstenar [3]:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Agila metoder erkänner möjlighet för kunden att kunna ändra sig under projektet gång. De inser också att utvecklingsproblem som kan dyka upp under projektet inte kan fullständigt förutsägas i förväg och problemets definition kan ej med goda resultat till fullo beskrivas och begripas innan arbetet. Här föredrar man snarare att arbeta med små interaktioner under vilka man producera någonting konkret som man kan leverera till kunden t.ex. efter varje sprint. Man föredrar också ett fungerande system över tunga dokumentationer, starka relationer till produktbeställaren samt goda team som kan snabbt ändra i kravet och implementera nya krav efter kundens önskemål. Detta arbetssätt kräver ändå en god konstruktion och överskådlig arkitektur. Dokumentation av mjukvarusystem som produceras, förblir viktigt, inte minst för att det blir enkelt att när än ändra i kravet utan dubbelt arbete samt att systemet blir relativt enkelt att underhålla för utomstående utvecklare som inte har varit med under utveckling av det framtagna systemet.

[1] *Inom Scrum, delas hela arbetet och dess tillhörande arbetsuppgifter i små delmoment under vilken en del av arbetet ska vara avklarad. Varje sådana delmoment kallas sprint*

b. SGDF

Self-governance developper framework [4], är ett ramverk som vi har använt oss av för att strukturera vårt arbete innan projekt start, under projektet och efter varje sprint för att uppdatera vårt produktbacklog. Innan projektet startades har vi gått igenom två faser: *preliminära aktiviteter* och *planeringsaktiviteter*. Under de två faserna har vi kommit överens om vilka krav vi ska implementera, vårt mål med sprint, konventioner kring kodhantering, samarbete via Git, produktbacklog och estimering av tidåtkomst för varje arbetsuppgifter samt bestämningen av dess prioriteter.

Under projektets gång har vi itererat de ovan beskrivna aktiviteter efter varje sprint, men vi har också enskilt antecknat varje dag vad vi har gjort och hur lång tid det tog att göra. Bland olika aktiviteter som vi har antecknat om i tidsrapportering, finns kodskrivning, testning av olika implementationer för att se om de följer kravet, evaluering av aktivitet samt felsökning. De anteckningarna har försett oss med en överblick på hur snabbt vi arbetar samt vilka arbetsuppgifter vi lägger mest tid på respektive mindre tid på.

I sista sprint hade vi som mål att leverera ett välfungerande spel som vi publicerade sedan på Google Play Store[2].

c. Essential Kernel

Essential Kernel [5] är ett ramverk som har tagits fram av Semat [3] (ett initiativ för att omforma mjukvaruutveckling så att programvaruteknik kvalificerar sig som en rigorös disciplin [6]). Essential Kernel använts främst och först för att hjälpa projektgrupper inom mjukvaruutveckling att veta hur projekt fortlöper, i vilket stadium de befinner sig i och vilka beslut som bör fattas för att föra projektet framåt till nästa stadium. Dessutom underlättar det framtagning av krav som på så sätt gör det enklare att följa projektets utveckling.

För att underlätta uppföljning av projektets status, har SEMAT tagit fram någonting som heter Alpha och varje Alpha har sina ”checklistor” eller underkategorier som bör uppfyllas för att projekts nuvarande stadium ska förflytta till nästa. Som bilden nedan illustrerar.

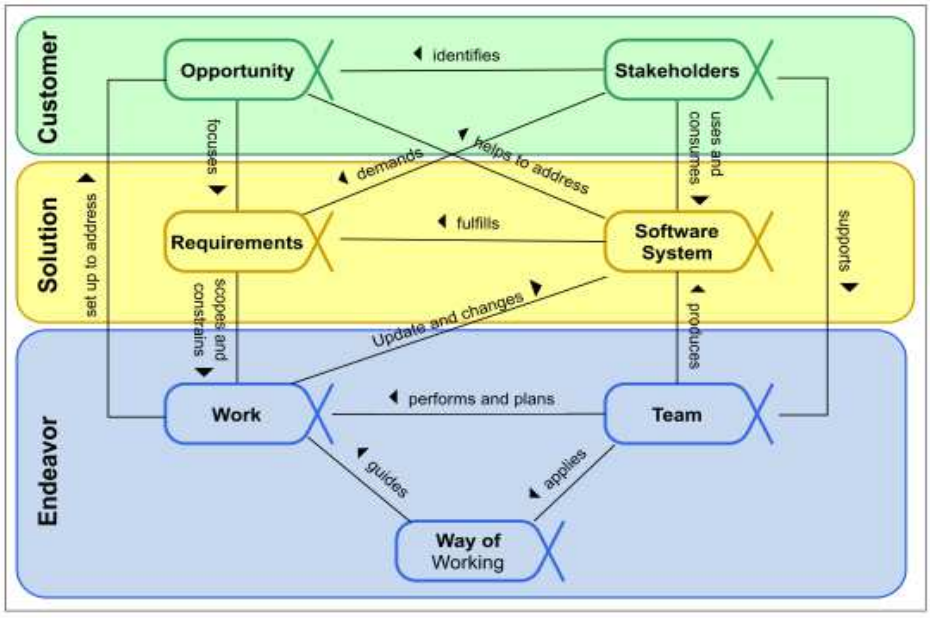


Fig. 1 Alpha

[2] *Bara tillgänglig för Android, under namnet Radiance*

[3] *Software Engineering Method and Theory , se http://semat.org/essence-user-guide*

Arbetssättet med mjukvaruutvecklingsmetoder som ett team

Som det har nämnts ovan, så har vi i vår grupp arbetat främst och först med 3 huvudsakliga arbetsmetoder: Scrum, Essence Kernel, och SGD. Scrum har varit det arbetssättet som vi har följt under hela projekt med sprintplanning, user stories, scrum master m.m.

Vår projektgrupp bestod av en Scrum master som såg till att arbetet följde Scrum under hela projektets livscykel. En produktägare som gav stora riktlinjer på hur produkten ska konstrueras samt vilka prioriteringar som bör göras vid varje sprint. Utöver det, hade projektgruppen flera utvecklare. I och med, det här projektet utfördes i samband med en akademisk kurs, så var både Scrum master och produktägare med och utvecklat produkten precis som alla andra deltagande studenter. Dessutom hade vi inga andra externa intressenter förutom testpersoner (mer om det, kommer senare).

Arbetsuppgifterna för projektet delades i fyra olika sprint. Innan första sprint, påbörjades ett förarbete där vi tillsammans diskuterade vilken produkt vi som grupp skulle ta fram. Efter att ha gemensamt kommit fram till den produkten som vi skulle ta fram, kom vi överens om «the definition of done» [4]. I vårt fall kom vi fram till att vi ansåg en task vara avklarad om och endast om den är implementerad, följer kravet samt går att demonstrera. Definition of done lämnades in till kursansvarig därefter började arbete med produktbacklog och krav.

Vi bestämde i generella termer den övergripande logik som vårt spel skulle ha samt vilka objekt vi skulle implementera och dess funktion i spelets övergripande system. De kraven skrevs ner, för att utgöra en user story [5], i ett Excel dokument som vi hade i molnet på Google Docs. De ”user stories” delades sedan i små task som vi skrevs ner på en postItlapp som fig. 2 visar. Efter har itererat samma process för alla objekt vi ville implementera, rangordnades alla uppgifter utifrån dess prioritet, efter en närmare bestämning om vilka som skulle vara med i den nuvarande sprinten. Prioritet bestämdes lite arbiträrt med generellt sett analyserade vi det underliggande beroende mellan olika uppgifter. De uppgifter som endast kunde implementeras efter andra uppgifter var avklarade hade lägre prioritet och vice versa.

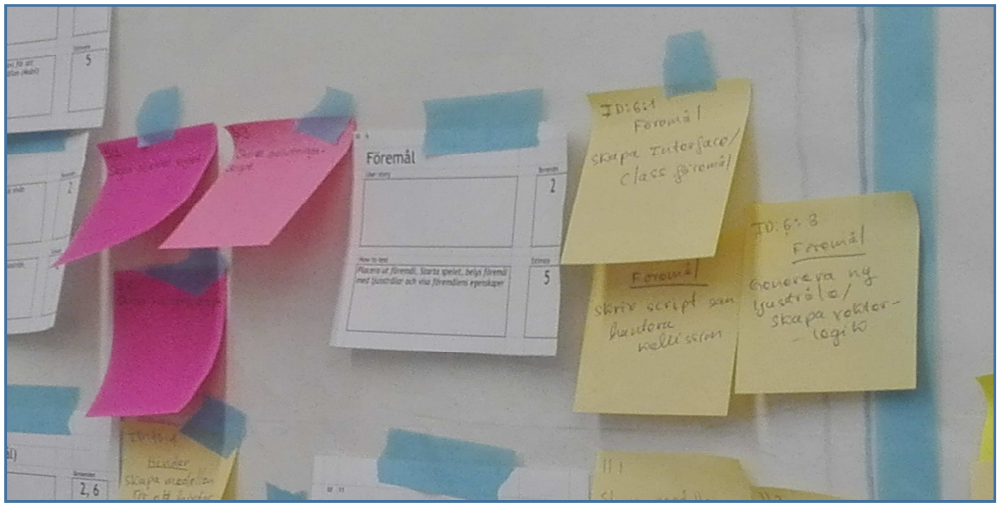


Fig. 2 task board

[4] *Definition of done, är ett begrepp som används bland annat inom Scrum för att veta exakt när en uppgift anses vara avklarad och implementerad.*

[5] *För varje ny objekt som implementeras skrevs en user story, som är en översiktlig beskrivning av vilka funktionalitet som det implementera objektet ska ha eller uppfylla.*

När arbetet med Sprintbacklog, ”user story” och dylikt avklarades, bestämde vi oss att översiktligt estimera hur lång tid skulle implementationen av varje ”user story” ta. För att kunna estimera i grupp och samtidigt ge möjligheten till alla medlemmar att tycka till om tidsestimering använde vi planning poker med hjälp av internetwebtjänsten pointingpoker[6]. Vi estimerade varje uppgift tills vi kom fram till en konsensus och detta sedan blev den officiella tiden som vi utgick från i vårt arbete under sprinten. Sedan bestämdes gemensamt målet med sprint.

Detta förfarande itererades i början av varje sprint fram till projektets slut. Vi hade däremot valt att inte använda matematiska metoder eller dylikt för att estimera om vår bedömda effektivitet (hastighet), som Henrik Kniberg förespråkar i hans bok [7]. Snarare hade vi valt att anta att vår fokus faktor kommer att hållas konstant och att vår estimerade effektivitet förblir oförändrad. Det valet har gjorts dels av brist på data för bättre estimering, begränsad tid för projekt (4 veckor) dels för att vi antog att arbetsförhållande kommer i stora drag att vara detsamma under hela projektet.

Utöver Scrum, har vi arbetat med Essentiel Kernel vars beskrivning har redan givits. Vi använde Essential Kernel för att veta hur vårt projekt utvecklas och vilka beslut vi bör fattas, vilka uppgifter som bör göras för att föra utvecklingen framåt. Som det framgick i fig. 1, så finns det totalt 7 Alpha. Vi delade ansvar för de olika alpha mellan varje medlemmar i gruppen, som hade plikten att följa utveckling av projektet utifrån det alpha som de hade ansvar för. Jag hade Alpha Work och resultat som vi kom fram till kommer att ges senare under Resultat.

Som det finns beskrivet ovan, använde vi också SDG för att veta exakt vad vår arbetstid gick ut på, vilka uppgifter som gjordes när och för hur lång tid. De data som vi har samlat i form av tidrapport finns att tillgå och resultat kommer översiktligt att ges under Resultat, alla andra data finns under bilaga. Dessutom användes Git för att underlätta arbetet mellan alla utvecklare, det användes för bättre kontroll över alla ändringar som gjordes i systemet (version kontroll) samt för att möjliggöra för oss att arbeta parallellt och samtidigt.

I vissa uppgifter använde vi också par programmering och efter varje sprint hade vi ett möte där vi diskuterade alla nya egenskaper som implementerats och varje utvecklare översiktligt berättade om vad de har arbetat med och hur deras kod och objekt fungerade.

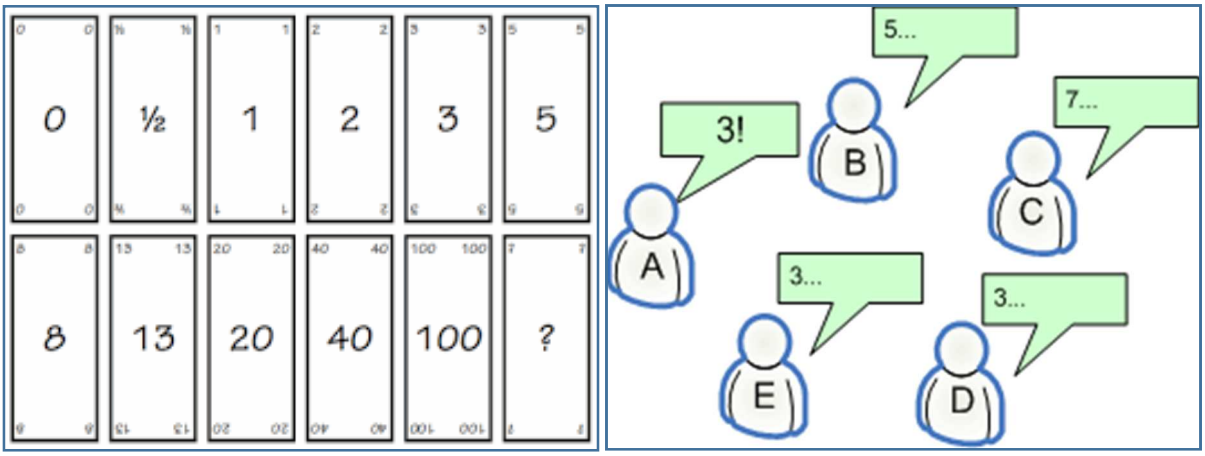


Fig. 3-4

[6] *https://www.pointingpoker.com/*

Resultat

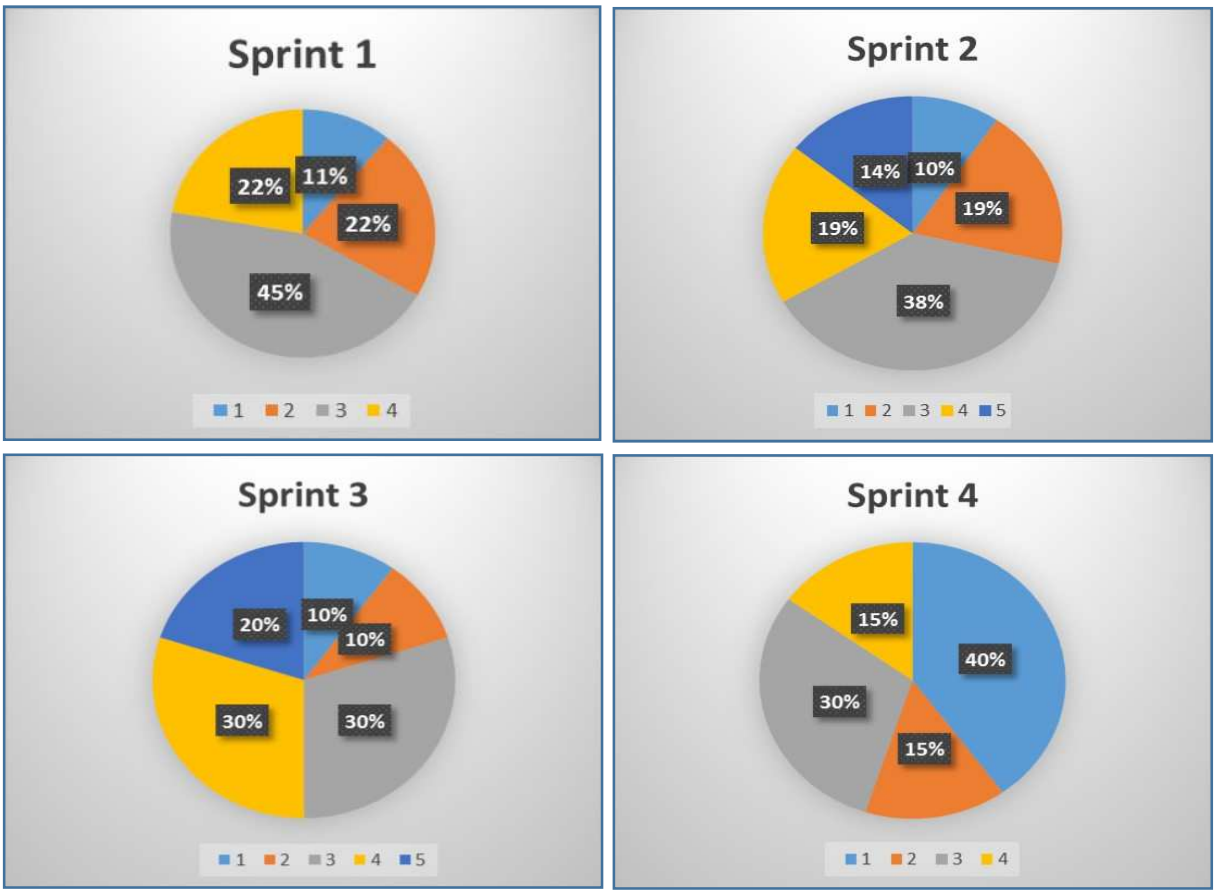


Fig. 5-8

Sprint 1

Under sprint 1, så gick stora delen av arbetet till att lära sig om samt bekanta sig med det nya verktyget. Det verktyg eller snarare plattform som vi använde för att utveckla spelet, var Unity. Mycket tiden lades på inlärning av den nya plattformen men samtidigt lades lika mycket tid på design såsom implementation av funktionalitet för de designade objekten via C# script i Unity, när vi blev tillräckligt vana med Unity. Fördelningen av arbetsinsatsen visas i Diagram 5 ovan där:

1. Miscellanous
2. Code writing
3. Analysis of the tool unity (Tutorial)
4. Game Design

Icke-utvecklarrelaterade aktiviteter:

Här arbetade vi mest med planering av första sprint: Vi satte upp vår tavla och delade in den i olika delar:

To do, in progress, next, unplanned, Sprint date. I todo, satte vi upp de uppgifter som bör utföras, In progress låg uppgifter som var under arbete och i next var uppgifter som vi skulle utföra om To-do lista blev tom, i unplanned satte vi upp bug och andra problem som dök upp. Vi planerade in också user story, möte m.m.

Utvecklarrelaterade aktiviteter:

Bland de tekniska delarna av arbetet under första sprint, så är det dem som finns redovisat i Diagrammet 5. Vi var helt nya till arbetet Scrum och Unity. Mycket tid investerades på att lära sig och resten av tiden gick ut på implementationen av våra högst prioriterade ”user stories” i Unity.

Sprint 2

I andra sprint, så blev vi lite mer insatta i hur Unity fungerar och hade tillräckligt förstått utvecklingsprocessen i Unity med C# script. Här fokuserades mest tid på att skriva kod och implementera logik för de objekten som skulle implementeras. Sedan hade vi många planeringsmöte för sprint och analys av kravet för ny införda objekt. Resten av tiden gick åt analysen av kravet, design, felsökning, som Diagram 6 visar

1. Bugfixing
2. Planning
3. Code Writing
4. Game Design
5. Analysis of requirement

Icke-utvecklarrelaterade aktiviteter:

Här den stora delen av vår tid gick åt möte för att skapa produktbacklog för sprinten: Diskussion kring kravet och andra objekt som skulle introduceras för att utöka användarupplevelse, tog också tid.

Utvecklarrelaterade aktiviteter:

I den här sprinten, utvecklade vi mycket. Vi skrev mycket skript för att utöka logiken i spelet och spelets komplexitet utökades därmed med nya objekt och häftigare logik.

Sprint 3

I tredje sprint, så hade vi som mål att leverera en beta-release av spelet, så vi arbetade därefter och fokuserad mest på skriva ny kod för att utöka logik och förebygga problem som kan dyka upp genom att skriva mer generell kod som kunde hantera alla andra subrutiner. Sedan tittade vi på olika bugar som fixades varefter de dök upp. Resten av tiden gick till analys av kravet, design och olika möte.

1. Analys av kravet
2. Planing och meeting
3. Felsökning
4. Code Writing
5. Design

Icke-utvecklarrelaterade aktiviteter:

Vi hade ofta möte för att förtydliga kravet för de nya objekt som skulle introduceras, sedan hade vi också planeringsmöte för att skapa nya produktbacklog samt diskutera spelets övergripande logik och struktur.

Utvecklarrelaterade aktiviteter:

I den här sprinten, utvecklade vi spelet vidare för att höja användarupplevelse. Vi fixade bugar som dök upp i spelet och utvecklade logik på så sätt att framtida objekt och script som skulle implementeras blev enkelt att integrera med resten av systemet.

Sprint 4

I fjärde sprint, hade vi planerat att ha en slutgiltig version av vårt spel som vi skulle lägga upp på Google Play Store. Så hela arbetet under sprinten gick ut på att förverkliga det målet. Stora delen av arbetet gick ut på att designa nya banor för spelet samt fixa de bugar som uppkom under tester. Dessutom lades en stor del av vår tid på att få in åsikter från olika testpersoner för att veta vad de tyckte och vad som skulle förbättras för att höja deras användarupplevelse.

1. Game Design
2. Code writing
3. Felsökning och bugfixing
4. Miscellaneous

Icke-utvecklarrelaterade aktiviteter:

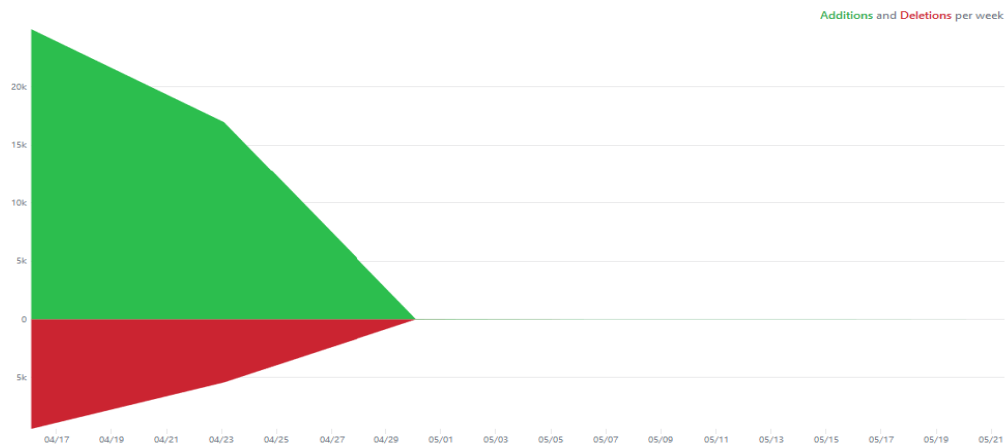
En stor del av arbetet lades på design av nya objekt, vilka texturer de skulle ha och vilket tema spelet skulle slutligen ha. Vi bestämde oss för rymd som temat. Dessutom lades en ganska stor del av tiden för samling av data från olika testpersoner. Vi utgick sedan från de framförda åsikterna för att ändra spelet så att användare skulle få en bra upplevelse.

Utvecklarrelaterade aktiviteter:

Stora delen av arbeta gick ut åt implementation av nya banor och användning av gamla objekt i Unity för att öka svårighetsgraden för spelare. Dessutom lades oerhört mycket tid på att fixa de olika buggarna som vi upptäckte under körningar och tester. Vi tittade också på hur prestanda och minnet skulle optimeras så att spelet skulle ta mindre plats hos användarens enhet.

Kodfrekvens i Git

Vi ska under analysdelen analysera hur vi har arbetat, de metoderna som vi har använt samt hur vårt arbete följer generella mjukvaruutvecklingsprocesser. Här nedan visas en kodfrekvens från Github för vårt projekt. Vi ska utgå från detta senare för att diskutera vårt projekt med Lehmans lag[7] som utgångspunkt.



Graf. 1

[7] *Det är en samling av lagar som Lehman stiftade för att generalisera utveckling av mjukvaror, se* https://en.wikipedia.org/wiki/Lehman%27s_laws_of_software_evolution

Alphautveckling eller projektutveckling över hela perioden

Under projektet, som nämnt ovan, hade vi arbetat med ett ramverk som heter Essentiel Kernel, det användes för att hålla reda på projektet status, det vill säga vilket stadium vi befann oss i vid varje tillfälle under vårt arbete. Alla data som vi har samlat kan hitta under bilaga. Dessutom på grund av utrymme har jag valt inte lägga upp olika grafer här, de kan också hittas under bilaga. I de olika graferna ser man utveckling av vårt projekt utifrån olika Alpha vid varje sprint, från den första sprinten till den fjärde och sista sprinten.

När det gäller Work, way of working och team alpha, så var de flesta av deras krav redan uppfyllda i mitten av projekten, de sista subalpha var uppfyllda enbart efter projektet, när vårt system levererades. Gällande Kund sektion, alltså Opportunity och Stakeholders, så var många saker inte direkt applicerbara på vårt projekt men en stor del av dem blev klara en bit innan projektets slut när vi fick in externa åsikter av fiktiva kunder[8] som skulle använda vår produkt.

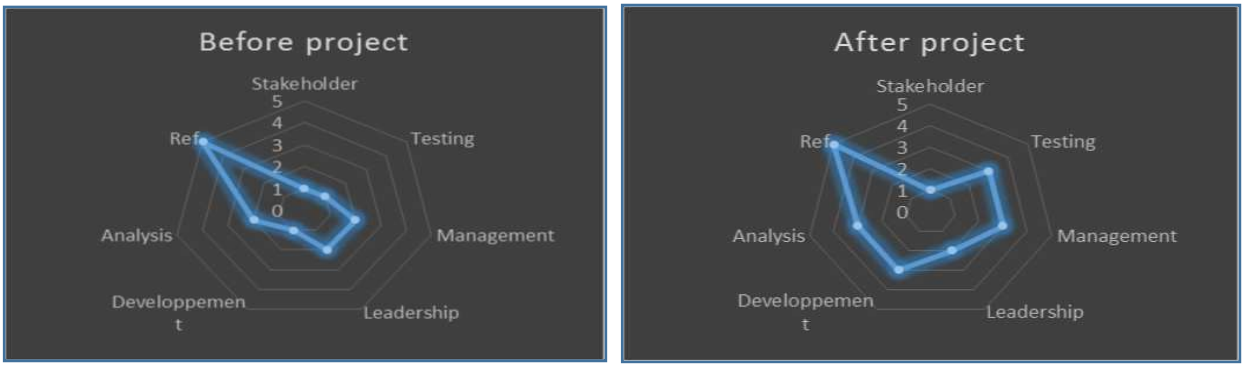
Till sist, fokuserade vi mycket mer på lösningssystem, Requirement och Software System. Det arbetade vi med under hela projektets, kraven utvecklades hela tiden, t.ex. vid varje sprint för att utveckla spelet vidare för på så sätt vidare höja värdet för våra intressenter. Dessutom utvecklades mjukvarusystem kontinuerligt för att underlätta hanteringen för oss utvecklare när komplexitet ökade samt för att göra spelet mycket mer effektivt när det gäller prestanda med också konkurrenskraftigt genom smartare logik via C# script. För mer detaljerad information om hur utveckling har sett hur under hela projekt se [5] och Bilaga.

Mätt personlig utveckling

I enlighet med SGD ramverk som vi har använt under hela vårt projekt, har varje utvecklare i projektgrupp mätt sina personliga egenskaper eller snarare sin kompetens innan och efter projektets genomförande. Den bedömningen gjordes utifrån 6 olika aspekter:

- ✓ Intressenter
- ✓ Testning
- ✓ Analys
- ✓ Utveckling
- ✓ Ledarskap
- ✓ Management

Grafen nedan vi visar den bedömning som jag har gjort på min egen kompetens innan samt efter projektets genomförande. Se Semat för mer information.



Graf. 2-3

[8] *Fiktiva kunder, i den bemärkelsen att de egentligen inte ska köpa produkten men snarare testa produkten för att höja dess värde genom vidareutveckling och förbättring*

Analys

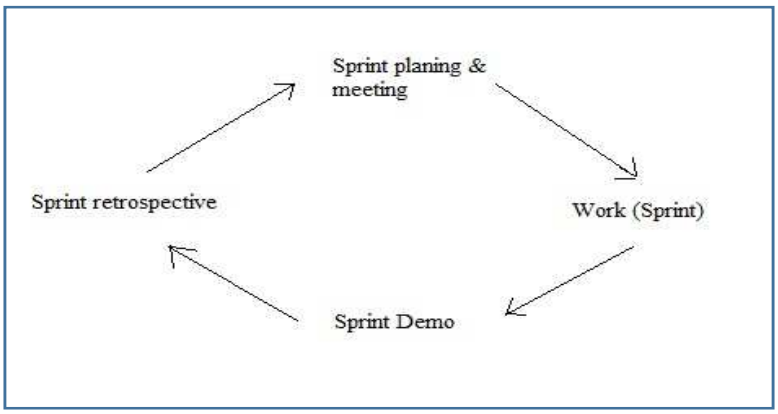


Fig. 9

Vi har arbetat under hela projektet med agila metoder, nämligen Scrum metodiken som det finns beskrivet i det agila manifestet [3]. Fig 9 sammanfattar den processen vi har använt och itererat vid varje sprint, under fyra veckoarbete. Vårt arbetssätt har varit väldigt anpassad efter den agila metodiken, dels för att vi har haft ett begränsat antal utvecklare i gruppen [10] dels för att vi har haft en regelbunden rytm vid varje sprint med sprint demo [9].

Om man analysera hur vår gruppsammansättning kan ha bidragit till eventuella framgång och missförhållande, kan man se starka samband, precis som det finns beskrivet i artikeln [8], där författarna har analyserat olika framgångsfaktor och missförhållande i olika mjukvaruprojekt för att hitta vilka faktorer som utgör god samt dålig praxis. Bland de faktorerna som bidrog till framgång i ett mjukvaruprojekt har de hittat, 4 olika faktorer som var starkt kopplat till ett lyckat projekt och bland de negativa faktorerna som bidrog till missförhållande fann de 4 olika anledningar som ledde till misslyckande, som tabell. 1 sammanfattar:

God praxis	**Dålig praxis**
1. Steady Heartbeat;	*1. Rules & Regulations driven;*
2. Fixed, experienced team;	*2. Dependencies with other systems;*
3. Agile (Scrum);	*3. Technology driven;*
4. Release-based (one application).	*4. Once-only project.*

Tabell. 1

Hur är det som tabell.1 visar kopplat till vårt projekt? Som jag precis nämnt ovan vid upprepade gånger så har vid arbetat agilt under hela projektet. Vi har haft ett gott projekt på så sätt att vi följt många av de goda praxis som förespråkas i artikeln [8]. Vi har haft en bestämd arbetsrytm med sprint demo efter varje sprint med samma tidsintervall [9], ett bestämt antal utvecklare i ett och samma ställe under hela projekt, vi har arbetat med Scrum och mot en enda applikation. Utöver det, vi har haft mindre beroende på andra system, få regler bortsett från sociala sådana. Vi har varit mindre teknologisk inriktad. Däremot har vårt projekt varit ett engångsprojekt och vi har inte riktigt använd en mer testdriven approach i vårt arbete, detta är några brister som kan åtgärdas i framtida projekt. I stora drag, har vi följt en god praxis.

I slutet av projektet, började vi med ta fram feedback och det var då vi började lite mer seriöst testa våra implementationer. Detta skulle kunna förbättras i framtida projekt, genom att hela tiden testa implementationer

mot kravet och funktionalitet, det vill säga innan varje implementation av kod eller objekt borde man ha en uppställning tester som den kod som implementeras ska gå igenom för att dels säkerställa att den levererar den kvalitet som vi eftersträvar delsförsäkra att den har rätt funktionalitet. Ett mer testdrivet arbetssätt vore att föredra i framtiden. Trots att vi inte ha haft kontinuerliga tester under projektet, så kan vi ändå analysera hur den tekniska utvecklingen av systemet har varit under de fyra sprint. För att göra detta kommer vi att använda samma förfarande som i [1] och jämföra vårt projekt utifrån Lehmans Lag [12].

Lehmans Lag är en samling av olika lagar, som är nuläget är sammanlagt 8 stycken. De lagarna beskriver olika faser som mjukvarusystem går igenom i ett typiskt föränderligt mjukvaruprojekt.

1. *(1974) "Continuing Change"*
2. *(1974) "Increasing Complexity"*
3. *(1974) "Self Regulation"*
4. *(1978) "Conservation of Organisational Stability (invariant work rate)"*
5. *(1978) "Conservation of Familiarity"*
6. *(1991) "Continuing Growth"*
7. *(1996) "Declining Quality"*
8. *(1996) "Feedback System"*

En fråga som nu kommer upp här, är nämligen hur utvecklingen i vårt projekt sett ut utifrån Lehmans lag. Det man kan säga är att utvecklingen har i många avseende följt de olika faserna som Lehmans lag beskriver. Om vi analysera varje lag för sig och gruppera dem i olika kategorier såsom i [1] kan vi bättre analysera. Lag 1 och 6 kan sammanfattas som förändring och utveckling, lag 3,4,5,8 som själv-reglering och feedbacksystem samt Lag 2 och lag 7 som stigande komplexitet respektive avstigande kvalitet [1].

Lag 1 & 6 förändring och utveckling

Vårt system har förändrats hela tiden under hela projektet gång, vi varje sprint har nya objekt introduceras och defekta eller mindre adekvata objekt har tagit bort precis som figur XXXXX kodfrekvens i Git visar. Här ser man utveckling följer Lehman utsägelser. Däremot när det gäller kontinuerlig tillväxt, precis som kodfrekvens visar de flesta script skrevs under sprint 1 och 2. I de senare sprinten introducerades väldigt få nya script, däremot har systemet växt med nya objekt och bibliotek istället.

Lag 3,4,5,8 självreglering och feedbacksystem

När det gäller självreglering och feedbacksystem, så har stabilitet i vårt arbete inte varit invariant under varje sprint, arbetsbörda har var högt under sprint 2 och 3, där avviker vi lite. Däremot har vi konstant underhållit en viss familjaritet med vårt system i gruppen. Varje utvecklare kunde relativt snabbt förstå vad andra har arbetat med. När det gäller själv-regleringen hade vi inte helheten i åtanke när vi började. Vi tog inte hänsyn till hur vi skulle underhålla vårt system när det växte med tiden. Men detta började vi åtgärda under sprint 3, där vi började skapa funktioner och script som kontrollerade alla andra subsystem, detta gjorde hanteringen och införande av nya funktionalitet hanterbart och överskådligt. Feedbacksystem påbörjades också lite sent, vid sprint 3. Vid framtida projekt bör de brister åtgärdas med till exempel tidigare tester, kontinuerlig feedback och överskådlig implementationer med helheten för det slutgiltiga systemet i åtanken.

Lag 2 Stigande komplexitet

Vårt system har självklart växt i komplexitet under hela projektet med nya objekt, logik och flera beroende mellan olika script och objekt som introducerades under tiden.

Lag 7 Avstigande kvalitet

Kvalitet på vårt system började försämras ju mer vi arbetade i gruppen, bland annat för att vi inte hade tillräckliga bestämmelse på hur saker skulle se ut samt att kravet var väldigt vagt beskrivet. Detta är en kompromiss som vi

har gjort, nämligen att inte ha tillräckligt bestämda krav för att behålla den flexibilitet som agila metoder erbjuder. Vid sprint 3 började se över våra konventioner för att få ett helhetligt och bra system som möjligt.

I framtida projekt, kan man se till att ha mer tydliga krav men tillräckligt mycket frihet för att ge utvecklarna den autonomin och flexibiliteten som agila metoder erbjuder men också att samtidigt att ge en väl beskriven mall och tydliga krav dels för att möjliggöra bygget av ett helhetligt system utan kompromiss om kvalitet dels för att motverka dubbelt arbete som missförstådda krav kan leda till.

Slutsats

Vad kan vi dra från slutsats från vårt arbete med Scrum? Vi har arbetat i en grupp av 8 studenter och det enda material som vi främst och först har använt för att bekanta oss med Scrum är boken skriven av Henrik Kniberg [7]. Scrum har varit väldigt intressant arbetssätt och väldigt anpassat för vårt projekt. Det gav oss flexibilitet att omforma projekt, ta in åsikter och anpassa projektarbete därefter utan stora dokumentationer och byråkratiska hinder. Den flexibilitet som Scrum ger, är någonting många företag letar efter. Möjligheten att när som helst under projektets gång ändra eller anpassa kravet efter nya omständigheter såsom tekniska hinder, tidsbrist eller kundernas önskemål. Vi också kunnat utnyttja den fördelen.

Dessutom har korta iterationer, hjälpt oss att se direkt resultat på det vi höll på att utveckla. Vi kunde samtidigt se små detaljer under planeringar men också ha helheten i tanken när vi skulle demonstrera vårt arbete efter varje sprint under sprint demo inför andra grupper. Detta har varit väldigt givande. Dessutom har vi utvecklat själva som individer men också som framtida ingenjörer. Vi har lärt också mer om hur det att arbeta som utvecklare i en större grupp med Git. Git har hjälpt oss att synkronisera och ordna vårt arbete så att vi dels kunde arbeta parallellt dels kunde vi bygga vidare på det andra har programmerat.

I kort, Scrum är ett bra arbetssätt som kan starkt rekommenderas även för nybörjare. Scrum tillsammans med Essence Kernel har varit till stor hjälp för att följa utveckling i vårt projekt. Detta är någonting som kan hjälpa alla utvecklare amatörer som professionella, i arbetsrelaterade projekt, i privata projekt såsom i akademiska sådana.

Referenser

[1] R. Sindhgatta, N. C. Narendra, and B. Sengupta, “Software Evolution in Agile Development : A Case Study ∗,” pp. 105–114.

[2] J. Campbell, S. Kurkovsky, and A. Tafliovich, “Scrum and Agile Methods in Software Engineering Courses,” pp. 319–320, 2016.

[3] http://agilemanifesto.org/

[4] M. Kajko-mattsson, “Self-Governance Developer Framework.”

[5] Q. R. Guide, “The Essence Kernel.”

[6] https://en.wikipedia.org/wiki/SEMAT

[7] H. Kniberg, *SCRUM AND XP FROM THE TRENCHES*, 2nd ed. InfoQ.com, 2015.

[8] H. Huijgens and A. Van Deursen, “How to Build a Good Practice Software Project Portfolio ?,” pp. 64–73, 2014.

[9] Don Wells, “A Project Heartbeat,” 2009. [Online]. Available: http://www.agile-process.org/heartbeat.html. [Accessed: 27-May-2017].

[10] J. E. Hannay, N.- Lysaker, and H. C. Benestad, “Perceived Productivity Threats in Large Agile Development Projects Categories and Subject Descriptors,” no. 1325. ------→

[11] S. Ilieva, “Adopting an Agile Methodology — Why It Did Not Work,” pp. 33–36.

[12] Wikipedia, “Lehman's laws of software evolutiont” 2016. [Online]. Available: https://en.wikipedia.org/wiki/Lehman%27s_laws_of_software_evolution. [Accessed: 27-May-2017]

Bilaga

A.

Här är en exempel på en av mina tidrapport. Alla fullständiga dokument hittas på https://kth.instructure.com/groups/7460/files/.

Activities

ACTIVITIES THAT I CONDUCT IN THE ROLE OF A STUDENT

Course 1: Time to fill “My Daily Work"

ACTIVITIES THAT I CONDUCT IN THE ROLE OF A SKILLED GENERALIST

Managing Requirements of a Skilled Generalist Role
REQ 1: Identify requirements
REQ 2: Analyze requirements
REQ 3: Change requirements
REQ 3.1: Identify impact of change (time it takes to identify changes everywhere in the system)
REQ 3.2: Make change(s) (time it takes to make changes everywhere in the system)
REQ 4: Plan requirements
REQ 4.1: Estimate effort
REQ 4.2: Prioritize requirements
REQ 4.3: Other planning activities related to requirements (specify)
REQ 5: Other, specify

Design of a Skilled Generalist Role
DES 1: Design system or system component (high-level design)'
REQ 2: Analyze requirements
DES 3: Change design
DES 3.1: Identify impact of change (time it takes to identify changes everywhere in the system)
DES 3.2: Make change(s) (time it takes to make changes everywhere in the system)
DES 4: Other, specify

Testing (NON-DEVELOPER Level Testing, acceptance, system, integration) of a Skilled Generalist Role
TEST 1: Define tests
TEST 2: Analyze tests
TEST 3: Change tests
TEST 3.1: Identify impact of change (time it takes to identify changes everywhere in the system)
TEST 3.2: Make change(s) to tests (time it takes to make changes everywhere in the system)
TEST 4: Other, specify

Project management of a Skilled Generalist Role
PROJ-MAN 1: Plan project/part of project (iteration)
PROJ-MAN 2: Analyze project/ project plan /part of project (iteration)
PROJ-MAN 3: Change project plan /part of project plan (iteration)
PROJ-MAN 4: Evaluate your project work
PROJ-MAN 5: Manage risks
PROJ-MAN 5.1: Identify risks
PROJ-MAN 5.2: Analyze risks
PROJ-MAN 5.3: Manage risks
PROJ-MAN 6: Customer-related activities, specify
ROJ-MAN 7: Other, specify

SGD ACTIVITIES CONDUCTed IN THE ROLE OF A (Pair)/DEVELOPER

Preliminary Activities
PR- 1: Review and agree on the overall or part of the project plan.
PR-2: Revise and ensure that the technology to be used is tested and understood.
PR-3: Revise and understand any appropriate internal (organizational) and external standard(s).
PR-4: Learn/relearn the organizational implementation and unit (developer) testing way of working.
PR-5: Review and revise your personal implementation and unit (developer) testing way of working.
PR-6: Other, specify

Planning Activities of a Developer Role

PL-1: Review the requirement(s) for the unit(s) to be developed.
PL-2: Prepare (make) and/or review the design specification(s) for the unit(s) to be developed.
PL-3: Resolve unclear questions and uncertainties.
PL-4: Determine and document your implementation and unit (developer) testing goals.
PL-5: Determine your implementation and unit (developer) testing strategy.
PL-6: Determine appropriate implementation and testing practices.
PL-7: Identify standards to be used for meeting your goals.
PL-8: Set your own personal deadlines to be met during your implementation and unit (developer) testing work.
PL-9: Estimate effort and resources required for carrying out your work.
PL-10: Schedule your work.
PL-11: Review your implementation and unit (developer) testing plan to ensure that it is realistic and achievable.
PL-12: Identify risks related to your plan.
PL-13: Plan for managing any identified risks.
PL-14: Other, specify

Preparatory Activities of a Developer Role

P-1: Prepare(make) and/or review your low-level design(s) of the code to be written or changed.
P-2: Prepare (make) an impact analysis of your low-level design(s).
P-3: Determine the types of unit (developer) test cases and their order.
P-4: Create and/or revise your unit (developer) test case base.
P-5: Revise the existing unit (developer) regression test base, if relevant.
P-6: Create or modify stubs and drivers, if required.
P-7: Prepare your unit (developer) testing environment and check whether it is appropriate for you work.
P-8: Other, specify

Coding Activities of a Developer Role

C-1: Write/rewrite your code.
C-2: Compile/ recompile your code as required.
C-3: Make notes on your compilation errors, if necessary.
C-4: Make notes on your defects
C-5: Other, specify

Unit Testing Activities of a Developer Role

T-1: Check whether the unit (developer) test case base meets the given requirements and design.
T-2: Check whether the unit (developer) regression test base meets the given requirements and design.
T-3: Remedy requirements problems in your unit (developer) regression and/or test cases base, if any.
T-4: Perform dynamic testing by executing code.
T-5: Perform static (human) testing by reviewing your code.
T-6: Record/write down test results.
T-7: Other, specify

Evaluative Activities of a Developer Role

E-1: Analyze your unit (developer) testing results.
E-2: Depending on the unit (developer) testing results, determine your next step(s).
E-3: Other, specify

Debugging Activities of a Developer Role

D-1: Identify the source of (an) error(s).
D-2: Determine solution(s) for eliminating the sources of error(s).
D-3: Other, specify

Self-Assessment Activities (Document aside your self-assessment results)

A-1: Assess your own development work.
A-2: Identify causes of your mistakes.
A-3: Identify improvement areas in your own way of working.
A-4: Other, specify

Delivery of a Developer Role

S-2: Deliver your code.

S-3: Other, specify

Preliminary Activities

PR- 1: Review and agree on the overall or part of the project plan.
PR-2: Revise and ensure that the technology to be used is tested and understood.
PR-3: Revise and understand any appropriate internal (organizational) and external standard(s).
PR-4: Learn/relearn the organizational implementation and unit (developer) testing way of working.
PR-5: Review and revise your personal implementation and unit (developer) testing way of working.
R-6: Sign your personal Service Level Agreement (Work Contract)

Planning Activities

PL-1: Review the requirement(s) for the unit(s) to be developed.
PL-2: Prepare and/or review the design specification(s) for the unit(s) to be developed.
PL-3: Resolve unclear questions and uncertainties.
PL-4: Determine and document your implementation and unit (developer) testing goals.
PL-5: Determine your implementation and unit (developer) testing strategy.
PL-6: Determine appropriate implementation and testing practices.
PL-7: Identify standards to be used for meeting your goals.
PL-8: Set your own personal deadlines to be met during your implementation and unit (developer) testing work.
PL-9: Estimate effort and resources required for carrying out your work.
PL-10: Schedule your work.
PL-11: Review your implementation and unit (developer) testing plan to ensure that it is realistic and achievable.
PL-12: Identify risks related to your plan.
PL-13: Plan for managing any identified risks.

Preparatory Activities

P-1: Prepare and/or review your low-level design(s) of the code to be written or changed.
P-2: Prepare an impact analysis of your low-level design(s).
P-3: Determine the types of unit (developer) test cases and their order.
P-4: Create and/or revise your unit (developer) test case base.
P-5: Revise the existing unit (developer) regression test base, if relevant.
P-6: Create or modify stubs and drivers, if required.
P-7: Prepare your unit (developer) testing environment and check whether it is appropriate for you work.

Coding Activities

C-1: Write/rewrite your code.
C-2: Compile/ recompile your code as required.
C-3: Make notes on your compilation errors, if necessary.
C-4: Make notes on your defects

Unit Testing Activities

T-1: Check whether the unit (developer) test case base meets the given requirements and design.
T-2: Check whether the unit (developer) regression test base meets the given requirements and design.
T-3: Remedy requirements problems in your unit (developer) regression and/or test cases base, if any.
T-4: Perform dynamic testing by executing code.
T-5: Perform static (human) testing by reviewing/inspecting your code.
T-6: Record/write down test results.

Evaluative Activities

E-1: Analyze your unit (developer) testing results.
E-2: Depending on the unit (developer) testing results, determine your next step(s).

Debugging Activities

D-1: Identify the source of (an) error(s).
D-2: Determine solution(s) for eliminating the sources of error(s).

Week 1										
Mon	People	Tue	People	Wed	People	Thu	People	Fri	People	**Total**
										0

				3H	8					**8**
		2H								**0**
										0
										0
		1H						1H		**0**
				2H						**0**
										0
										0
										0
										0

										0
				4H						**0**
										0
										0
										0
										0

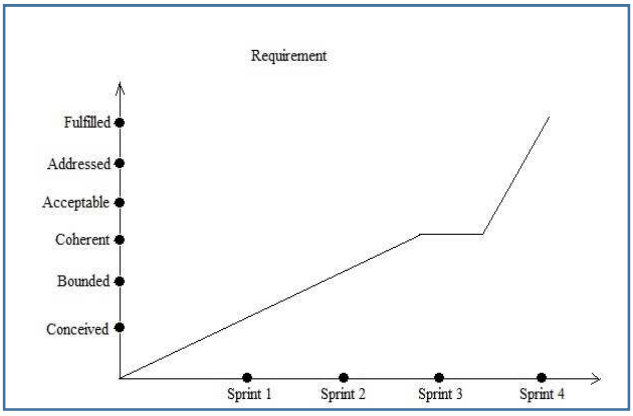
										0
										0
								2H		**0**
										0
										0
										0

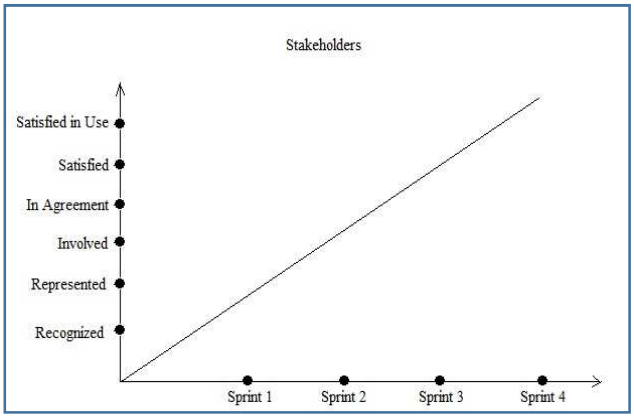
plan for the new sprint t = 3h										**0**
					3	3H				**3**
										0
										0
					1	1H				**1**
										0
										0
										0
										0
										0

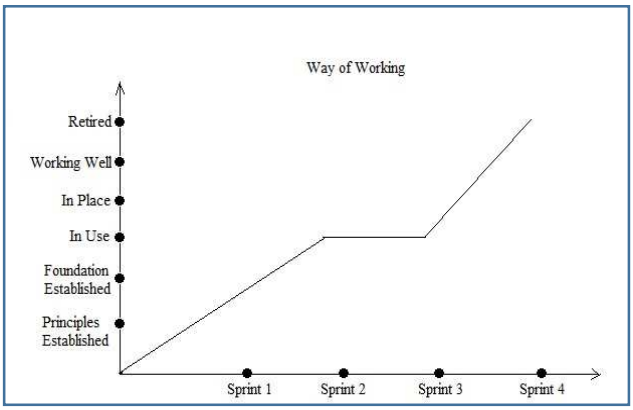
Work										**0**
								3H		**0**
										0
										0
										0
										0

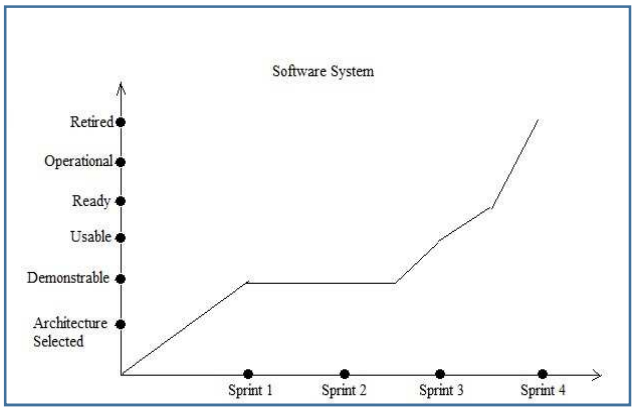
			2H	2H	0 0 0 0 0 0 0 0 0 0 0 0 0 0
			2H		0 0 0 0 0 0 0 0 0
	2H				0 0 0 0 0
write code for one feature 4h					0 0 0 0 0 0 0 0
					0 0 0
	2H				0 0 0
identify mistake					0 0 0 0
					0

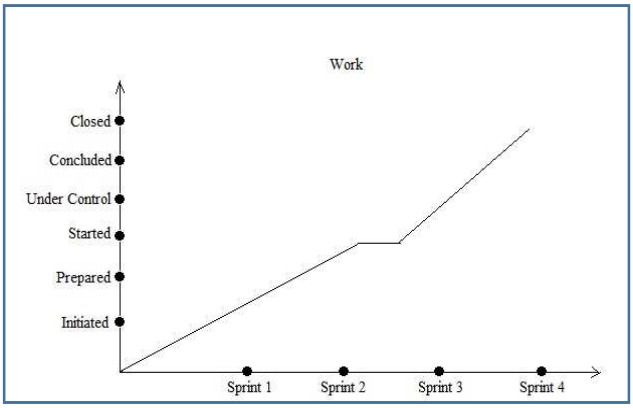
B. *Alphautveckling under varje sprint*

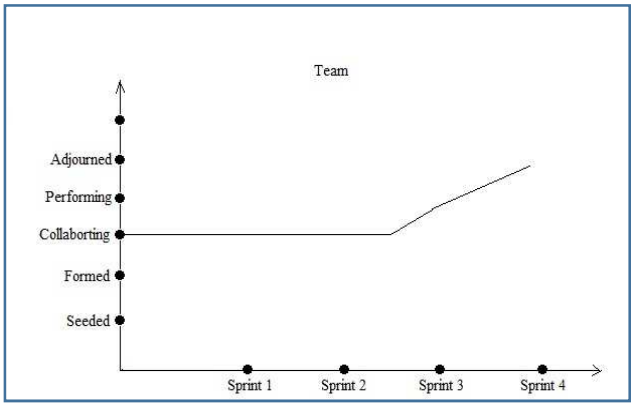


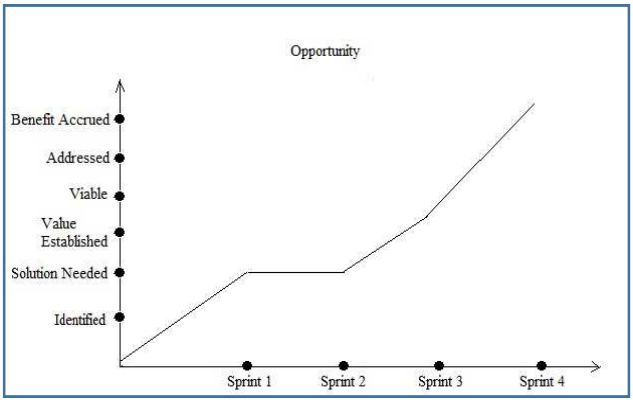












C.