Individuell Slutrapport
PA1416 : Programvaruprojekt i grupp

Blekinge Tekniska Högskola, Maj 2018

Grupp : 6

David Hörnmark (dhornmark@gmail.com)

Kund : NKT

Fredrik Ripa (fredrik.ripa@nkt.com)

Fredrik Pettersson (fredrik.pettersson@nkt.com)

1. Introduktion

1.1. Information gällande projektet

Målet med detta projekt var att skapa ett djupare implementerat Proof of Concept (för att enkelt beskriva det, applikationen i sig är inte i ett stadie då det kan tas i drift och ersätta den nuvarande arbetsprocessen i och med att det behöver leva upp till deras ISO-standarder) för min kund, NKT. NKT var intresserade av ett protokollföringsystem som skulle hjälpa testarna med att fylla i och spara datan för de prov som de har utfört på deras högspänningskablar.

Projektet fick namnet Cavedo som är en sammansättning av de två första bokstäverna i orden Cable, Verification och Documentation. Cavedo var även ett av NKTs första steg till en digitalisering inom företaget.

Fördelen med Cavedo gentemot metoden som testarna använder sig av nu är att testarna med hjälp av Cavedo inte behöver skriva in informationen flera gånger. Tidigare var de tvungna att först fylla i datan på ett protokoll i papper, sedan ett Excel-ark för den interna lagringen av datan och slutligen ett dokument som blir det provprotokoll som kunden får. Cavedo har samlat denna funktionalitet i en webbapplikation där testaren endast behöver fylla i data för den kabel som används, välja relevanta provtyper och fylla i dess data en gång för att sedan kunna generera PDF:er av protokollet. Detta effektiviserar arbetsprocessen för testarna och minskar risken för mänskliga fel i och med att antalet gånger som datan manuellt behöver skrivas in har sänkts från 3 gånger till 1 gång.

För Cavedo valde jag och mitt team att använda oss av Model-View-Controller arkitekturen samt att bygga systemet som en webbapplikation. Model:en består utav en SQL Server 2016-databas från Microsoft. View:n är uppbyggd av ASP.NET, C Sharp (C#), HyperText Markup Language (HTML), Cascading Style Sheets (CSS), JavaScript, Asynchronous JavaScript and XML (AJAX), jQuery, Bootstrap och ASP.NET Razor. För en vanlig användare ser det ut som en vanlig hemsida. Controller:n använder sig av teknikerna ASP.NET, C Sharp (C#) och Entity Framework för att skicka information mellan Model:en och View:n.

1.2. Information gällande kunden

De som har varit våra huvudsakliga kontaktpersoner och stakeholders på NKT har varit:

- Fredrik Ripa (fredrik.ripa@nkt.com)
    - Manager High Voltage Routine Test Laboratory
    - Direktnummer: +46 455 559 53
    - Mobil: +46 705 695 641
- Fredrik Pettersson (fredrik.pettersson@nkt.com)
    - Test Engineer at High Voltage Routine Test Laboratory
    - Direktnummer: +46 455 33 41 72
    - Mobil: +46 725 440 307

1.3. Information gällande mig själv

Mitt namn är David Hörnmark och är nu i slutskedet av min fjärde termin på programmet International Software Engineering här på BTH. Jag har genom tidigare kurser i mitt program fått kunskap inom programmering i flertalet programmeringsspråk såsom C++, Java, JavaScript, HTML, SQL och CSS. Den kunskap som har varit relevant för detta projekt är det jag har lärt mig i kurserna:

- **PA1444 Webbprogrammering och databaser**
    - I denna kurs lärde jag mig att skapa webbapplikationer (HTML, JavaScript, CSS m.fl.) och hantera databaser i MySQL (som använder SQL som språk).

- **DV1540 Inledande programmering i C++ och DV1537 Objektorienterad programmering i C++**
    - Tack vara dessa två kurser så hade jag en bra bas gällande att förstå hur objekt och "pratar" med andra objekt och hur jag ska göra för att skicka med information i t.ex. parametrar. Det har ju också varit till hjälp att C# (som användes i projektet) är så nära besläktat med C++.

- **PA1414 Individuellt programvaruprojekt**
    - I mitt individuella projekt så utvecklade jag en webbapplikation till min kund Christian Lundmark. Denna applikation var ett automatiskt bevattningssystem som man kunde via webbapplikationen.
    - Detta individuella projekt gav mig möjlighet att använda mina kunskaper som jag lärt mig i PA1444 och finslipa dem ännu mer.
    - Det har varit betryggande att ha denna kunskapen med mig under utvecklingen av Cavedo.

- **DV1549 Datastrukturer och algoritmer**
    - Det jag fick utav denna kurs var en djupare förståelse för hur man designar bra funktioner, sett utifrån tidskomplexitet, och har varit förmånligt att ha under projektet då det har krävts att man skriva lite komplexa funktioner för att bearbeta datan som skickas.

- **DV1557 Användbarhet och interaktionsdesign**
    - I och med att jag var ansvarig för Front-end:en på Cavedo så har DV1557 varit till stor nytta.
    - Det jag lärde mig i denna kurs var hur man designar en användbar applikation. Användbarhet utgörs utav aspekterna Förmågan att lära sig applikation, applikationens effektivitet, förmågan att komma ihåg hur en applikation fungerar även om det var länge sen man använde den, felhantering och den subjektiva tillfredsställelsen.
    - Denna kurs läste jag nu i LP1 i termin 4. Jag blev färdig med den i en väldigt lämplig tidpunkt för det var i samma stadie som att vi skulle börja göra GUI:et “användbart”.

2. Resultat av projektet

De krav på funktionalitet som bestämdes i kontraktet var:

- ➢ Användaren ska kunna lägga till ett protokoll
- ➢ Användaren ska kunna ogiltigförklara ett protokoll
- ➢ Användaren ska kunna redigera ett protokoll
- ➢ Användaren ska kunna parkera ett protokoll
- ➢ Användaren ska kunna återuppta ett parkerat protokoll
- ➢ Användaren ska kunna lägga till en provhall
- ➢ Användaren ska kunna inaktivera en provhall
- ➢ Användaren ska kunna redigera en provhall
- ➢ Användaren ska kunna lägga till en utrustning
- ➢ Användaren ska kunna inaktivera en utrustning
- ➢ Användaren ska kunna redigera en utrustning
- ➢ Användaren ska kunna signera dokumentet vid färdigställning
- ➢ Användaren ska kunna exportera det färdigställda dokumentet
- ➢ Användaren ska kunna generera ett löpnummer
- ➢ Användaren ska kunna söka efter protokoll, provhallar, utrustningar och tester i databasen
- ➢ Applikationen ska innehålla ett användarvänligt GUI

Detta var också den funktionalitet som vi i gruppen levererade. Det blev inga omförhandlingar eller byten av krav. Vi levererade inte heller något extra.

Något som jag i efterhand nu ser är att dessa krav är väldigt vaga. Det finns alldeles för mycket utrymme för tolkning för att jag ska vara bekväm med det. Det är en av de saker som man lär sig av att ha denna typ av projekt.

Så som jag har förstått det så är alla funktioner och Features tillfredsställande, detta i och med att med har fått acceptans av produkten.

Som jag och gruppen tidigare har nämnt i Grupprapporten så hade vi inte någon bra struktur på loggning av hur mycket tid som vi lade på de specifika Features:en. Detta medför att vi har en total summa timmar som är arbetade gentemot den totala summan som vi har planerat för. Detta är absolut inte optimalt, men man får se det som att hade vi kunnat allt det här med planering och grupprojekt sedan innan så hade väl knappast denna kursen givit oss något nytt.

Status	ID	Krav / Beskrivning	Budget	Estimering
	F1	Lägga till ett protokoll i databasen via webbsidan	10	64
	F2	Ogiltigförklara ett protokoll från databasen via webbsidan	4	25.6
	F3	Redigera ett protokoll som finns i databasen via webbsidan	8	51.2
	F4	Parkera ett protokoll till databasen via webbsidan	5	32
	F5	Återuppta ett parkerat protokoll från databasen via webbsidan	4	25.6
	F6	Lägga till en provhall i databasen via webbsidan	5	32
	F7	Inaktivera en provhall från databasen via webbsidan	2	12,8
	F8	Redigera en provhall som finns i databasen via webbsidan	4	25.6
	F9	Lägga till en provtyp i databasen via webbsidan	8	51.2
	F10	Inaktivera en provtyp från databasen via webbsidan	3	19.2
	F11	Redigera en provtyp som finns i databasen via webbsidan	5	32
	F12	Lägga till utrustning i databasen via webbsidan	3	19.2
	F13	Inaktivera utrustning från databasen via webbsidan	1	6,4
	F14	Redigera utrustning som finns i databasen via webbsidan	2	12,8
	F15	Generera nummer för ett nytt protokoll via webbsidan	2	12,8
	F16	Ange den data för kabeln som används i protokollet via webbsidan	2	12,8
	F17	Signera protokollet när det är färdigt via webbsidan	1	6,4
	F18	Exportera ett färdigt protokoll till en PDF-fil via webbsidan	10	64
	F19	Söka efter och visa protokoll som finns i databasen via webbsidan	1	6,4
	F20	Webbsidan ska ha ett användarvänligt GUI	20	128
		TESTNING/BUGFIX		400
		Totalt:	100	1040

Här har vi alla Features och dess budgetering, tyvärr finns som sagt ingen rapporterad tid tillgänglig för de individuella Features:en.

Tid för kurs	Tid för Features / Testning	Förbrukad tid för projektet
1600 timmar	1040 timmar	1268.5 timmar

Enligt figuren ovan så kan vi se de totala timmarna i sitt sammanhang. Anledningen till att vi inte hade en bra struktur på den administrativa sidan var för att vi såg inte riktigt de långsiktiga vinsterna i det. Nu i efterhand förstår åtminstone jag vikten av att ha allt loggat och

dokumenterat. Att ha en bra spårningsförmåga i ett projekt är nyckeln till att snabbt kunna hitta de punkter som behöver förbättras eller korrigeras.

Sprint	Planerad tid	Rapporterad tid	David	Elias	Mirza	Emmie	Vecka
1	128	114	8	9	9	9	3
			20	20	20	20	4
2	128	130	12,5	11	14,5	12	5
			20	20	20	20	6
3	128	134	20	20	20	0	7
			18	20	16	20	8
4	128	115	16	20	12	15	9
			12	12	10	18	10
5	128	233	26	4	26	25	11
			36	38	40	38	12
6	128	171	18	18	32	20	13
			20	20	23	20	14
7	128	98.8	6	10	8,5	8	15
			16	24	8	18	16
8	57.6	132	12	22	15	15	17
			20	18	14	16	18
9	86.4	140	13	13	0	10	19
			17	20	8	14	20
			12	12	12	9	21
Total:	1040	1268.5	322,5	331	308	307	

Enligt denna tabellen (tagen ut vår grupprapport) så kan vi se att mina totala timmar var 322,5.

3. Lärdomar

3.1. Projekt i mjukvaruutveckling

Man kan utföra projekt i mjukvaruutveckling på flera olika sätt. Men oavsett vilken metod man väljer så utför man stegen (nödvändigtvis inte i denna ordningen):

1. Samla krav från kunden.
2. Skapa en design
3. Implementera enligt designen
4. Testa designen funktionellt
5. Få feedback från kund, användare eller stakeholder detta kan även ses som icke-funktionell testning.
6. Installation av produkten.
7. Underhåll av produkten.

Alla metoder utför inte dessa steg på samma sätt. En sekventiell metod utför varje steg en gång medans en iterativ metod kan utföra flera steg flera gånger. En agil metod liknar mycket en iterativ metod men skillnaden är att i en iterativ metod så blir man inte färdig med en Feature i ett svep utan man gör en iteration om och om igen tills man är färdig medans man inom agil utveckling levererar en färdig produkt vid varje slut av en sprint. Vid användning av en sekventiell metod så får kunden inte en leverans av en färdig produkt förrän projektet är slut. Det är också svårt då att göra ändringar till skillnad från agil utveckling där du kan göra ändring ofta i och med att leveranserna är fler och oftare.

En av de mest vanliga är Vattenfallsmetoden (Sequential / Lifecycle). Det finns olika versioner av Vattenfallsmodellen men om man ska bryta ner det så består modellen huvudsakligen av dessa sju viktiga steg:

1. Kravspecifikation för systemet.
2. Design av systemet.
3. Implementation av systemets komponenter.
4. Integration av systemets komponenter.
5. Verifiering av systemet.
6. Installation av systemet.
7. Underhåll av systemet.

Enligt denna modell så skall man vara helt färdig med ett steg innan man går vidare.

En av de mest kända iterativa metoderna är Prototyping. Prototyping liknar väldigt mycket Vattenfallsmetoden, skillnaden är att man här är beredd att göra ändringar när det krävs för att göra kunden nöjd. Detta gör ma genom att iterera om tidigare steg som Kravspecifikation, Design av systemet och konstruktion. De steg som då existerar i Prototyping är:

1. Övergripande kravspecifikation för systemet.
2. "Snabb" design av prototypen.
3. Konstruktion av prototypen (antingen bred eller djup implementation)
4. Utvärdera prototypen och förbättra kraven.
5. Skapa den färdiga produkten, enligt vattenfallsmetoden, utefter prototypen.

En iteration kan ske vid varje skede i stegen (rimligast är ju 2 - 4) så fort ett fel eller ett annat krav har dykt upp. När man sedan är nöjd med prototypen så kan man fortsätta enligt vattenfallsmetoden.

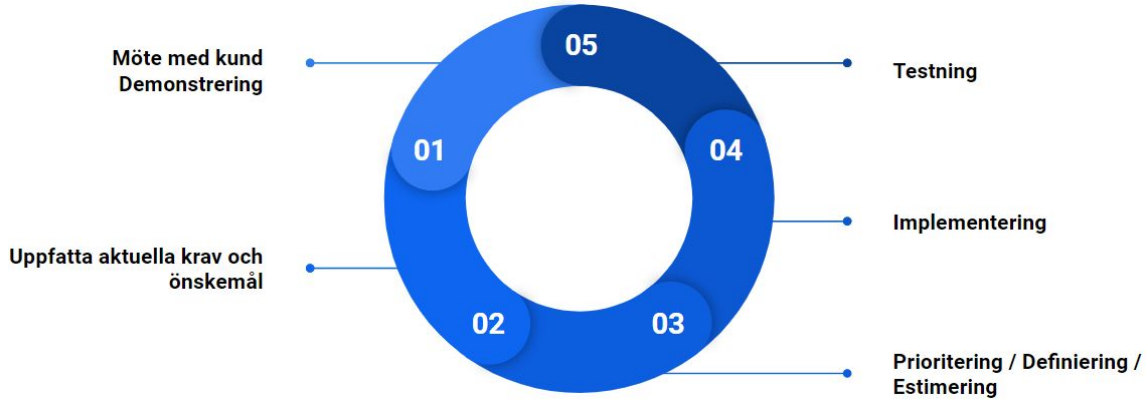
Slutligen har vi då den agila utvecklingen som tar en annan approach. Här arbetar man i sprintar där man utför alla dessa stegen av att finna kraven och utveckla något som är färdigt för stunden.

Enligt det agila manifestet[1] så förespråkar man inom agil utveckling:

- Individer och interaktioner över processer och verktyg.
- Fungerande mjukvara över omfattande dokumentation.
- Samarbete med kunden över förhandling av kontrakt.
- Att anpassa sig till förändring över att strikt följa en plan.

Här ser man ganska tydligt skillnaden mellan agil och sekventiell utveckling. T.ex. den sekventiella utvecklingen följer strikt en plan som man tidigare har förhandlat fram.

I vårt team försökte vi att arbeta enligt Scrum, men på grund av att Features:en inte var nedbrutna tillräckligt så var de för mycket sammankopplade för att man skulle kunna leverera en fungerande komponent efter varje sprint. Istället så tog projektet en approach mer liknande Prototyping. Vi skapade tidigt "skalet" för systemet och sedan med varje sprint så utvecklade vi det "skalet" djupare och djupare i och med den feedback vi fick från mötena med kunden. Strukturen av vårt arbete blev på så sätt ganska iterativt med en struktur som bestod av stegen:



Denna figur ska tolkas som att vi började med att ta reda på vad kunden verkligen ville ha och att vi sedan implementera det som vi tror tar oss närmast det. Därefter visar vi det för kunden och de får komma med feedback på hur vi kan komma närmre den bilden som de hade. Slutligen då i sista sprinten så hade vi en produkt som både vi och kunden kände var nära det de ville ha.

3.2. Kravanalys, kommunikation med kund och erbjudande

A. Enligt kurslitteraturen[2] på sidorna 62-65, så börjar varje projekt med att man identifierar syftena (aims) och målen (objectives) med själva arbetet. Efter att man har specificerat vad man vill uppnå med projektet så kan man börja med att göra målen SMART:a där S står för Specific, M för Measurable, A för Appropriate, R för Realistic och T för Time-Related.

   Genom att göra målen/kraven SMARTA gör att man faktiskt kan genomföra och bekräfta att de är färdiga. Ett mål som inte är mätbart kan man inte heller bekräfta när det är uppnått.

B. I min grupp arbetade vi med krav genom att fråga vad kunden vill ha, skriva ner det, skapa en antingen en fysisk eller imaginär bild av det och bekräfta att vi har samma uppfattning. Vi hade ju även möten varannan vecka för att bekräfta att vi har uppfattat det de ville ha. Det är lättare att bekräfta om något är rätt när vi i gruppen faktiskt har utvecklat något som de kan se och känna på.

   Jag har lärt mig vikten att av kommunicera med kunden och att jag verkligen måste arbeta för att säkerställa att jag och kunden har samma uppfattning om vad som ska utvecklas. Till nästa projekt tar med mig insikten av att det är extremt viktigt att man gör kraven SMART:a. Vi gjorde inte det i detta projektet och det är en stor risk skulle jag säga, såhär i efterhand. Som tidigare nämnt så krävs det ju att man kan mäta kraven för att verkligen kunna bekräfta att man uppnått dem.

Att göra erbjudandet var inte speciellt svårt i vårt fall. Jag tycker att kunden har varit väldigt förstående och samarbetsvillig. Det gör dock att jag förstår hur illa det kan vara om man har en kund som inte är förstående och samarbetsvillig. En lärdom som jag tar med mig är att alla vill ha så mycket som möjligt för så lite som möjligt och det är viktigt att man inte säljer sig för billigt och tar på sig mer än vad man kan hantera.

Kontraktsmässigt skulle jag poängtera en gång till att kraven hade behövt vara bättre, annars tycker jag att det har varit bra.

3.3. Planering och kontrollering

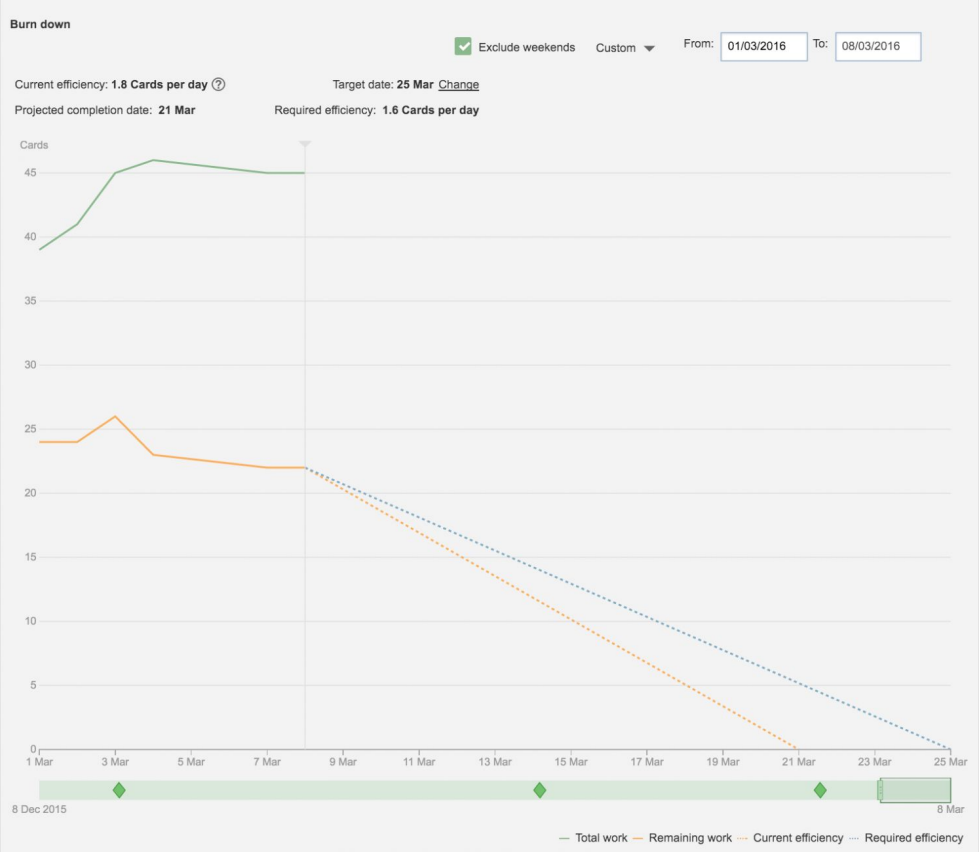
A. Enligt kurslitteraturen[2], på sidorna 65-80, så hjälper planering och estimeringar mig att förstå vilket arbete som måste utföras, vilken ordning som jag ska hålla på med arbetet och hur lång tid som jag behöver för varje uppgift.

   T.ex. tack vare denna överblick så kan jag snabbt se om projektet är för komplext eller om det saknar djup. Jag kan då åtgärda detta genom att antingen öka eller minska scope:t för projektet för att sedan planera om projektet.

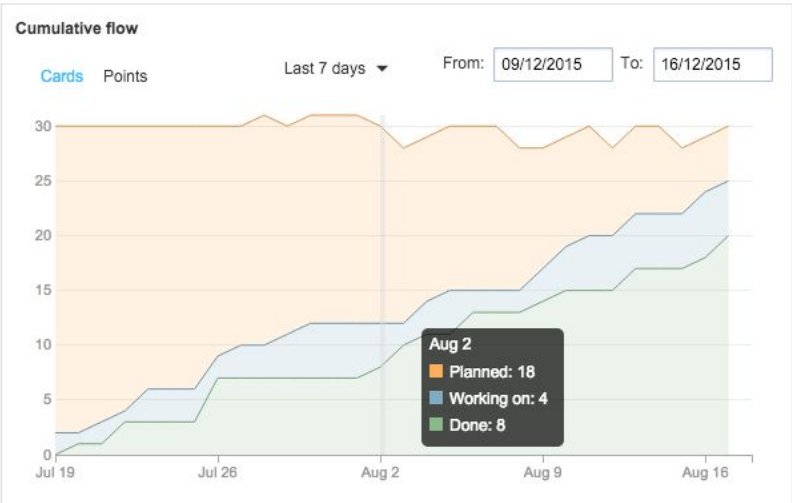
   Anledningen till att man följer upp på hur mycket tid man lägger på all uppgifter är för att man ska kunna omvärdera och flytta resurser till de uppgifter som behöver det. Denna förflyttning av resurser är kanske inget som gruppen själva kan göra och då är det viktigt att man har en bra Follow-Up så att de som tar besluten kan vara uppdaterade på projektets status.

   För t.ex. en Sprint så kan man använda flera olika metoder för att visa progressen. Figurerna nedanför är lånade från hemsidan Planview[3].

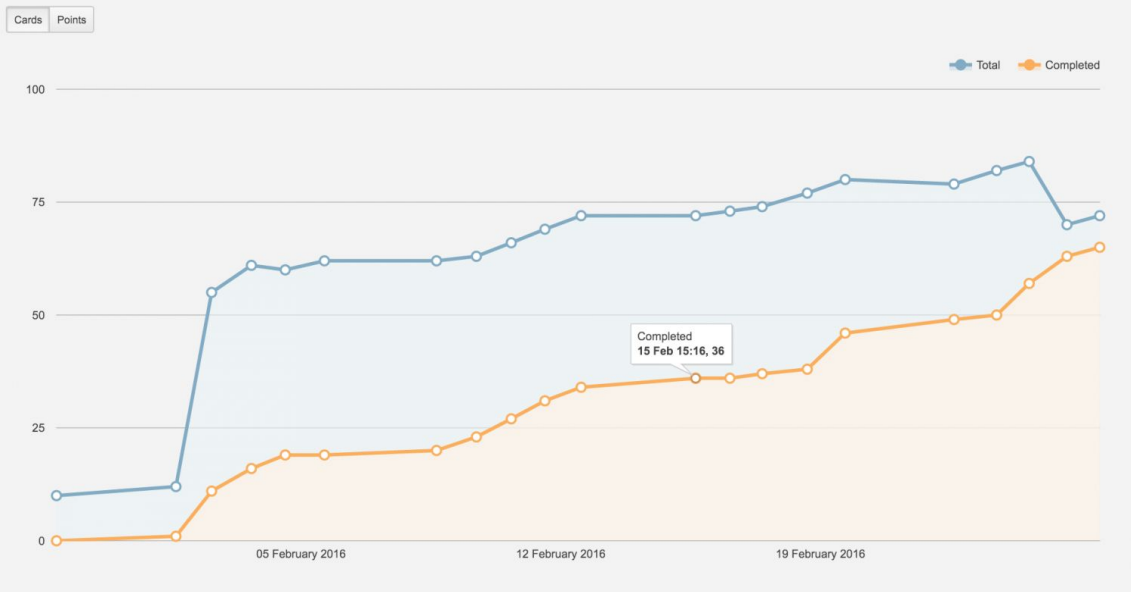
En typ av approach för att kommunicera progressen för projektet är att göra Burndown charts. Här betar man av aktiviteter allt efter som och drar av deras “poäng” från totalen. Målet är att nå 0 för varje sprint. Denna metod är mer inriktad för en individuell sprint.



En annan metod för presentera progressen för hela projektet är att använda sig av Cumulative Flow Charts. Dessa presentera en mer hel bild av progressen för projektet där man kan se alla planerade aktiviteter, aktiviteter som är i gång och de aktiviteter som är färdiga.



Utöver dessa två så finns motsatsen till Burndown alltså Burnup chart. Det följer samma princip som Burndown fast målet är att samla poäng så att man når totalen av alla planerade poäng.



Med hjälp av alla dessa metoder så kan man se när saker inte sker i den takt som det ska och på så sätt kan man i ett tidigt skede tillsätta de åtgärder som krävs.

B. I min grupp gjorde vi så att vi började med att använda Dollar metoden för att prioritera de Features som vi hade kommit överens med kunden om. Vi hade 100 Dollar som vi splittade mellan alla Features. Dessa prioriteringar utfördes endast av oss. En lärdom att ta med till nästa projekt är att låta kunden vara den som skriver prioriteringarna. Vi hade ju kommunikation med kunden om vad de ville ha mest men för att göra det mer formellt så hade man kunna låta kunden göra sina prioriteringar. Dessa prioriteringar översatte vi sedan till planerade mantimmar genom att göra en multiplikation av 6.4. Anledningen till att vi valde den siffran var för att vi kände att den gav en balans mellan totalt planerade timmar för projektet och den tid vi trodde att de individuella Features:en skulle kosta.

Som jag och gruppen har nämnt tidigare i Grupprapporten så förde vi ingen bra loggning på hur mycket tid en specifik Feature kostade. Detta medför att jag kan inte göra en jämförelse mellan vilka aktiviteter som var estimerade fel. Vi använde estimerade heller inte om några Features. Detta för att vi var så inne i att bara leverera projektet, så vi såg inte vinsten i att hålla på med det administrativa. Detta kan ha berott på att vi var en liten grupp (fyra personer) hade vi varit flera så hade vi insett vikten av att ha allt dokumentar så att alla kan hållas informerade. Överlag skulle jag dock säga att de flesta Features:en

var estimerade fel. Anledningen till detta var för att vi hade i princip ingen erfarenhet av att utveckla i de tekniker som valdes, att veta hur mycket tid något skulle kosta var extremt svårt. Med erfarenheten som jag fått av detta projekt så kommer jag att kunna göra bättre estimeringar i framtiden.

Ordningen som vi skulle utföra aktiviteterna styrdes av den huvudsakliga funktionalitet med Cavedo, att kunna Skapa ett protokoll och fylla det med kabeldata och testdata. Vi började då alltså med att se vad vi behövde för att göra det möjligt och vi matchade då Features och Sprintar på detta sätt. Jag skulle säga att vi gjorde det i rätt ordning och vi missade ingen aktivitet eller Feature.

Det enda som jag direkt ser att planeringen och Follow-Up:en hade kunnat hjälpa vårt projekt mer är kopplat till det stora beslutet vi behövde göra angående hur vi skulle använda tekniken för att skicka data mellan View:n och Model:en via Controllern. Hade vi kunna planera och bryta ner Features och planera för hur vi skulle skicka data:n så hade vi inte behövt göra en massa dubbelt arbete. Det blev dubbelt arbete för att vi hade gjort en byggt en del på systemet och sedan var vi tvungna att skrota det för att göra och göra rätt. En incident som denna hade även varit bra att kommunicera vidare via någon Follow-Up. Vi valde dock att bara köra vidare för vi alla i gruppen var införstådda med vad som hade hänt.

3.4. Tillämpade teknologier

Cavedo fick arkitekturen Model-View-Controller då vi ansåg att det var den mest lämpliga. MVC var mest lämpligt för att vi skulle arbeta mycket med databasen och informationen där och ville hålla den skiljd från Vy:n. MVC är även en arkitektur som används ofta i dagens programvara så vi kände att det fanns gott om information att tillgå ifall vi behövde hjälp.

För detta projektet så använde jag och gruppen mig av en mängd olika teknologier och program. Programmen vi använde var:

- Visual Studio 2017
- SQL Server Management Studio 17.7
- Git GUI
- Git BASH

Själva mjukvaran vi utvecklade bestod av:

- ASP.NET MVC Framework
    - Model:en använde sig av
        - SQL Server 2016.
    - Controller:n använde sig av:
        - ASP.NET
        - C#
        - Entity Framework 6
    - View:n (Front-End:en)
        - ASP.NET
        - C#
        - HTML
        - CSS
        - JavaScript
        - AJAX
        - jQuery
        - Bootstrap (för CSS)
        - Razor

Jag har personligen endast arbetat med HTML, CSS, och JavaScript förut så jag har under projektets gång fått lära mig att använda en massa nya teknologier. Det jag har fått lära mig är:

- Skicka data:n mellan View:s och Controller:n
- Skicka data:n till Model:n från Controller:n.
- Uppdatera databasen med den nyinkomna data:n
- Jag har även fått lära mig bygga hemsidor som består av Partial View:s och Modal:s.
    - Partial Views används på så sätt att jag skriver koden en gång och sen kan jag kalla på den som en byggnadssten i de andra CSHTML-dokumenten. Väldigt användbart när man ska t.ex. göra labels och input-lådor som ska ha specifik data från Model:n.
    - Modal:s är rutor som poppar upp när man t.ex. vill bekräfta att man vill radera en artikel. Fungerar liknande Partials men är mer temporär för så fort man är färdig med den så försvinner den. Partials stannar hela tiden.
- Jag har lärt mig använda Git, det var inget som jag använt i någon större utsträckning innan.
    - Lärde mig både använda Git BASH och Git GUI.
- Lärde mig även använda SQL Server Management Studio 17.7 för att hantera databasen.
    - Var bra för att kolla upp så att värden sparas i databasen samt för att korrigera saker för att underlätta testning. Jag kunde t.ex.ändra Bool-variabler i syfte för testning.

3.5. Testning och försäkran av kvalitet

Totalt så allokerade vi i gruppen 400 timmar till testning. Den faktiska totalen av timmar som vi faktiskt la är inte loggad så jag kan inte ge ett svar på hur mycket tid vi la på det. Anledningen till detta är som jag nämnt tidigare att vi insåg inte vikten av att logga allt.

Vad jag dock kan säga att vi testade det vi hade skapat innan varje demo-visning för kunden. När vi kör vårt demo så vill vi ju bygga förtroende så vi testade alltid hela GUI:t och dess delar innan visning. Självklart testar man det man kodat innan man pushar upp det. Så det sker testning på ens lokala branch och sedan sker integrationstestning efter man har satt ihop komponenterna.

Jag skulle med detta vilja säga att vi åtminstone hade bra acceptanstestning, enda negativa var att vi inte hade några mätbara krav så fick vi gå efter om kunden tyckte om det eller inte.

Slutligen så testade vi hela systemet innan slutleverans samt en del testning efter installation, för att korrigera eventuella fel som uppstått i samband med installationen.

3.6. Riskhantering

A. Enligt kurslitteraturen, på sida 80 - 86, så är Riskhantering en process som sker parallellt med projektplaneringen för att förbereda sig på samt går igenom hur man ska hantera negativa händelser som kan ske under projektets gång. Riskhantering sker i fyra olika steg:
   1. Identifiera riskerna
   2. Bedöma graden av påverkan riskerna kan ha
   3. Lindra de mest kritiska risker
   4. Kontrollera/hantera riskerna

   Anledningen till att man använder sig av Riskhantering är att varje projekt utsätts för risker och när en risk blir verklighet så ska gruppen inte stå utan en plan för hur incidenten ska lösas. Om gruppen inte vet hur det ska lösa problemet så kommer gruppen inte kunna jobba sig framåt och bli klara med projektet inom den budget som specificerats. Det viktiga är också att man fokuserar på de risker som har hög sannolikhet att inträffa i kombination med att de medför grova konsekvenser och gör upp en plan för hur man ska hantera dem.

B. I min grupp så hade vi i början av projektet en ganska oklar metod för hantering av våra risker. Detta berodde mycket på att vi inte hade lagt en massa arbete på att identifiera några risker vilket i sin tur berodde på att vi inte hade insett vikten av det administrativa.

Efter första review:n och möte med våran Head of Department, Nina Dzamashvili Fogelström, så lärde oss hur viktigt det var att faktiskt ta i tag med hanteringen av risker. När vi satte igång med riskhanteringen märkte vi att vi faktiskt redan hade kommit en bit på vägen då vi väldigt informellt hade identifierat riskerna om t.ex. vår begränsade kunskap inom teknikerna.

Resultatet av vårt arbete med att identifiera risker blev:

1. Begränsad kunskap inom teknikerna.

Denna risk identifierades genom att vi helt enkelt insåg att vi visste inte hur projektet skulle byggas, vi hade ingen historik eller erfarenhet av de främsta teknikerna C#, ASP.NET MVC Framework och Entify Framework.

2. Personlig skäl och sjukdom.

Denna risk identifierades genom att vi utsattes för en förlust av en gruppmedlem i ett tidigt skede av projektet.

3. Att våra estimeringar inte var korrekta.

Denna risk uppstod som en följdeffekt av vår begränsade kunskap.

4. Risker med kunden.

Det är viktigt för oss att kunna presentera det vi gjort under sprinten och få feedback på det. Lyckas vi inte få det så riskerar vi att jobba mot fel riktning. Det var också en risk att kunde inte kunde tillhandahålla oss med nödvändig hårdvara och mjukvara för att kunna möjliggöra livetester och installationer på plats.

5. Kontakt med slutanvändaren.

I och med att Cavedo byggs med hjälp av slutanvändare på så sätt att de är med på våra möten och får säga vad de vill ha och hur det ska vara så är det viktigt att vi har tillgång till dem. Hade vi inte haft det så det varit en stor risk för att vi kanske inte levererar “rätt” produkt i slutändan.

Vi som grupp hade inte gjort upp någon plan för hur vi skulle hantera våra risker. Vi arbetade bara till steget att vi identifierade dem, vilket så här efterhand kan tycka låter bristande. Det var bara så att vi inte visste att vi skulle behöva det. Vi gjorde bara det vi trodde var mest lämpligt, vilket i detta fallet var börja jobba så snabbt som möjligt för vi måste prova oss fram (i och med vår bristande kunskap inom teknikerna).

3.7. Verktyg och tekniker som användes

För dokumentering av krav så använde vi inga verktyg. Jag skapade ett dokument i Google Docs där vi samlade allting. Min reflektion kring detta är att det inte längre kan anses vara optimalt. Ett system som hade tillåtit oss att göra SMART:a krav hade varit bättre. Jag tror att ett av problemen till att vi inte hade så bra dokumentation var just för vi använde inga verktyg. Hade vi använt verktyg så hade vi kanske inte känt att det administrativa arbetet var något jobbigt.

Samma sak gällde estimering och planering. Allt vi hade för detta var samma dokument som vi använde för att dokumentera kraven, därmed tror jag också att hade vi använt verktyg här så hade vi haft en bättre översikt och bättre allmän koll. Vi hade säkert kunnat möjliggör omestimeringar och på så sätt planera projektet bättre. Till nästa projekt ska jag definitivt använda fler verktyg.

Vi använde GIT för att hantera våran källkod. Jag har tidigare inte använt det men jag tyckte att så fort man hade lärt sig det så fungerade det väldigt smidigt. Utan GIT hade det varit ett väldans trassel att försöka sätta ihop allas delar, när man ska MERGE:a kod manuellt så finns det en stor risk för mänskliga fel.

För Build så byggde vi bara projekt med den inbyggda funktionen Build Project i Visual Studio 2017 och för Release så användes den inbyggda funktionen Publish, i Visual Studio 2017.

För testning så hade vi byggt Unit Tester för databasen som man manuellt kunde köra i Visual Studio 2017. Vi använde oss inte av någon automatiserad testning. För större projekt kan det nog vara en fördel att köra automatiserade tester över nätterna. För testning av GUI:t körde vi manuell explorativ testning med en hel del error guessing. Det var dock väldigt informellt i och med att jag utförde den mesta så rapporterade jag aldrig vidare till någon annan. Detta är absolut något som måste ändras till nästa projekt. Jag hoppas på att kunna använda mig av en mer formell testning där det sker bra dokumentering över hela projektet.

Som reflektion känner jag att hade vi använt oss utav mer verktyg så hade det underlättat det administrativa arbetet och vi hade haft ett mer väl strukturerat och dokumenterat projekt. Jag tror även att det hade gjort testningen mer effektiv och bättre. Själva anledningen till att vi inte använde några en massa verktyg var nog för att vi inte direkt kände till några och vi hade ett

sådan tunnelseende av att bara komma igång och koda så resultatet blev att vi missade det totalt. Finns inget jag kan göra i detta skede mer än att ta med mig den lärdomen till nästa projekt.

4. Personliga erfarenheter

Jag skulle säga att en av de största utmaningarna för projektet har varit att arbeta tillsammans i en grupp. Det krävs att man måste verkligen jobbar för att bli effektiva tillsammans. I vårt fall löste vi det genom att vi fick roller som vi var bekväma med. På så sätt tror jag att vi undvek mycket frustration.

En annan utmaning har ju framför allt varit teknologierna. Vi (åtminstone jag) i gruppen hade som sagt ingen direkt tidigare erfarenhet av de primära teknologierna (C#, ASP.NET MVC och Enitity Framwork) så vi missade många steg i projektplaneringen just på grund av vi var så benägna att bara komma igång och koda så vi kan lära oss teknologierna. Vi var tvungna att testa oss fram och se vad som fungerade och vad som inte fungerade. Detta går även sedan in på punkten om vad som har varit de avgörande momenten för projektet. Ungefär halvvägs in så var vi valde vi att byta sätt som vi använde för att skicka data mellan View:n och Model:en. Detta var en markant ändring i och med att det innebar att vi gick från ett sätt vi va bekväma med men som var ineffektivt och rent av felaktigt (sätt ifrån tanken om hur man ska använda ASP.NET). Detta är vad jag refererar till när jag menar att vi tidigare var tvungna att göra dubbelt arbete. Resultatet av detta var att det med tiden underlättade det vårt arbete avseendevärt, då vi blev väldigt mycket mer effektiva i implementationen av Features..

Ett annat tillfälle som verkligen definierade vårt projekt var när vi bestämde mer fasta roller. Tanken från början var att vi alla skulle jobba med allt genom att rotera positioner. Detta insåg vi efter några veckor att det var alldeles för ineffektivt och problematisk för att det var verkligen svårt för oss att sätta oss in i någon annans arbete när man själv inte kunde tekniken. Vi löste detta genom att helt enkelt sätta fasta roller.

Den främsta lärdomen som jag tar med mig är att det krävs att man som grupp planerar och bryter ner projektet rejält. Att hitta alla byggnadsstenar som kommer att användas och ta reda på hur man ska sätta ihop det hela. Att köra en brute-force metod där man testar sig fram kommer med högsta sannolikhet inte fungera på ett storskaligt projekt. Anledningen till att jag känner att detta är av yttersta vikt är för jag har nu insett att en bättre planering hade hjälpt oss att ha koll på projektet och hade minska risken för att gruppmedlemmar arbetar med Features som inte är relevanta för tillfället (t.ex. de går inte att integrera för de resterande komponenterna inte är utvecklade).

Jag tar även med mig lärdomen om hur viktigt det är att skapa SMART:a krav och mål i specifikationen för systemet. Det är viktigt att man i kontraktet har krav som man kan mäta så att

man i en tvist kan bevisa att krav antingen har eller inte har uppfyllts. Inget ska lämnas till att tolkas. Jag har genom projektets gång förstått hur annorlunda olika människor kan uppfatta information. Jag och en kund kan använda samma ord men vi har olika definitioner för det och uppfattar det således på olika sätt. Viktiga dokument och begrepp skall vara definierade för att undvika detta problem.

Utöver dessa två lärdomar så tar jag med mig att det är en väldigt bra idé att i ett tidigt skede i projektet hitta verktyg för det administrativa arbetet som behövs göras. Anledningen till att jag känner att detta är viktigt är för att jag vet att det hade hjälp oss i vårt projekt med att ha koll på vad som måste göras och hur effektiva vi är samt visa vår progress. Ju mer vi lyckas avlasta oss och automatisera desto bättre.

En av de sista men säkerligen en av de viktigaste lärdomarna är vikten av att strukturera upp testningen. För detta projekt hade vi endast informell testning där inget dokumenterades. För mer komplexa system så kommer inte detta funka, så för det avancerade grupprojektet så kommer det krävas att det blir mer formellt och strukturerat. För nu hade vi inte så bra koll på vad som har testats och vilka fel som uppstått osv. Vi gjorde mest så att vi utförde vår explorativa testning och fixade de fel vi hittade längs med vägen. Detta går även in på det om att använda verktyg, hade vi haft verktyg för att utföra t.ex. UI-testningen så hade det underlätta väldigt mycket och vi hade som resultat fått ett mer pålitligt system.

För mitt nästa projekt så vill jag implementera bättre och mer grundlig planering av arbetet. Det är något som inte bara jag utan hela gruppen och projektet i sin helhet kommer att ha nytta av. Jag känner verkligen att det var något som saknades i detta projektet. Nog för att vi levererade det vi skulle, det är bara det att det hade kunnat gå smidigare.

Jag skulle även vilja implementera bättre testning. Mer strukturerad och formell (det betyder dock inte att jag vill exkludera informell testning) för att verkligen se till att man lever upp till de SMART:a krav som har specificerats. Mer strukture är också något jag känner att hela grupper har nytta av.

Slutligen vill jag säga att vägen må ha varit lite krokig men vi har nått fram tillslut. Våra brister har blivit synliga och jag känner att åtminstone jag har lärt mig en hel del. Jag ser fram emot nästa projekt där jag kan få visa allt jag lärt mig i denna kursen.

Vill även ge ett ytterligare tack till alla inblandade som har givit mig möjligheten till detta projekt.

5. Referenser

[1] Manifesto for Agile Software Development (2018/05/31)
http://agilemanifesto.org/iso/sv/manifesto.html

[2] Projects in Computing and Information Systems: A Student's Guide Second Edition, CHRISTIAN W. DAWSON, 2009, ISBN 978-0-273-72131-4
http://libris.kb.se/bib/11688259

[3] Follow up on work progress, Planview (2018/05/31)
https://success.planview.com/Projectplace/Execution_and_follow-up/Follow_up_on_work_progress