# XP – eXtreme programming:

EXtreme programming er en måde man kan arbejde på. Denne metode kom frem i starten af 90’erne. Det var en mand ved navn Kent Beck, der udviklede denne agile metode, der både omfatter planlægning samt udvikling af et produkt. Det der ligger i eXtreme, er, at der hele tiden lægges vægt på den sunde fornuft i det ekstreme. Det vil sige, at hvis man har fundet frem til, at test er godt, så skal der laves tests – hele tiden! EXtreme programming kan beskrives ved et sæt af værdier og praktikker. Der er fire værdier, og det er ting man værdsætter igennem et sådan forløb. De fire værdier er: Kommunikation, forenkling, tilbagemelding og mod. (måske mere uddybende?) Ud over dette er der som sagt også nogle praktikker, man kan anvende. Der er tolv praktikker, hvilket er: Planning game, kort tid mellem releases, metaforer, simpelt design, definér test-first, refactoring, pair-programming, kollektivt kodeejerskab, løbende integration, 37 timers arbejdsuge, kundeinvolvering samt kodestandarter. Disse praktikker vil blive uddybet senere. På figuren til (højre/venstre) kan man se, hvordan alle praktikkerne hænger sammen. Fjerner man én af praktikkerne, går det ud over kvaliteten af det endelige resultat. Praktikken ’Planning game’ går ud på, at man i samarbejde med kunden finder ud af, hvilke funktionaliteter produktet skal indeholde. Det gør man ved hjælp af ’User stories’. User stories laver man ved hjælp af et kort, der indeholder fire punkter, nemlig: En titel, et ’As a: ””’, et ’I can: ””’ og et ’So that: ””’ felt. En User story kan man definere som en slags ’klump’ funktionalitet, der giver værdi for kunden/brugeren. Derfor er det ideelt set kunden, der skal udfylde user storien, for programmørerne derefter kan estimere de forskellige Stories. En måde man kan estimere på, er ved at bruge noget, der hedder ’Planning Poker’. I Planning Poker er der tretten forskellige kort. Hvert medlem af teamet får et sæt af de tretten kort. Ud af de tretten kort er der elleve værdier: 0, ½, 1, 2, 3, 5, 8, 13, 20, 40 og 100. Disse værdier bruges til at estimere hver eneste User Story, ved at hvert medlem vælger ét kort med en værdi. Når medlemmerne har valgt et kort, viser de, hvad de har valgt og bliver derefter i fællesskab enige om, hvilken værdi Storien skal have. Ud fra de User stories der er lavet, findes den der menes at være en ’middel’-Story. Denne ’middel’-Story bliver estimeret med én af de elleve værdier og ud fra denne, bliver de andre User Stories estimeret. Efter dette kan kunden evt. opdele de forskellige stories op i mindre dele. Når det er gjort, kan man lave eksperimenter, også kaldet ’spikes’, i ukendte områder.

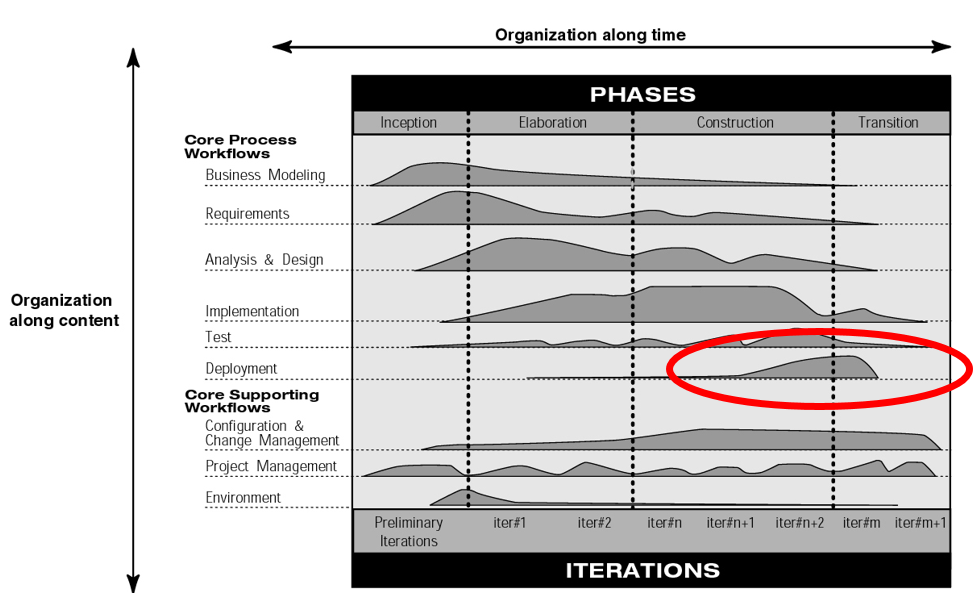
# Deployment:

Deploymentfasen er der, hvor et produkt skal gøres klar til at komme ud på markedet, og eventuelle kunder kan begynde at benytte sig af det. Men før man overhovedet kan snakke om at udgive eller lave en release på et produkt, skal man have udført en endelig accepttest, også kaldet User acceptance teststing (UAT). Udover UAT bliver der udført en masse små accepttests, når man, som tidligere nævnt, er i gang med at gøre de enkelte User stories færdige. UAT er som tidligere nævnt den sidste fase i udviklingsprocessen af produktet/softwaren. I denne fase tester man softwaren for at tjekke op på, om al funktionalitet er, som det skal være. Måden man tester dette på, er ved at give softwaren til nogle personer inden for den målgruppe, man har tænkt sig at ramme. Softwaren der skal testes, er som regel gratis i perioden, hvor den testes. Enten kan man lægge softwaren ud på nettet, så folk udefra kan teste det. En anden måde er ved at finde mulige brugere af produktet, og lade dem teste det på et internt system. Grunden til at det er vigtigt med denne UAT, er, fordi man kan få feedback på eventuelle fejl i softwaren, der skal rettes, før den bliver offentliggjort. Når man kigger på frigivelsen af produktet, laver man en alfaudgivelse af det. Alfa er den første af to slags tests. Her har man oftest et internt team til at teste dette samt potentielle brugere af softwaren. Når softwaren menes at have al ønsket funktionalitet, og det virker, udgives en det nu i en betaversion af produktet. Her laves en form for ekstern accepttest af produktet, hvor en større gruppe personer kan benytte sig af det. Produktet er stadig ikke 100 % færdigt, og alle mulige bugs skal findes og laves.

Inden man udgiver produktet helt, er der flere andre ting, ud over accepttests, der skal være styr på, hvis meningen med det, er at lægge det ud på markedet. Når det skal udgives, er det vigtigt, at man har en plan over, hvordan man har tænkt sig at markedsføre sit produkt. Denne plan indeholder f.eks.: En markedsføringsstrategi, en prissætning, hvilke salgskanaler man skal bruge samt eventuelle licenser. Ud over dette skal der også være styr på, hvor systemet skal ligge henne. Skal det ligge lokalt på en server, eller skal det gøres ved hjælp af en server på nettet. Til sidst skal man have sidste aftaler på plads, hvad angår domæner og licenser.

## UP og agilt syn på deployment:

Hvis vi starter med at se på UP’s syn på deployment, så gælder det om, at have lavet et workflow-skema, så der er lagt en plan for det hele, før man går i gang. Ved at lave sådan et skema, kan man lave en releasedato på produktet På denne måde kan man på forhånd aftale med kunden, eller fortælle de kommende brugere, hvornår produktet er klar til at benytte. Som man kan se på billedet nedenfor, så bliver deployment-aktiviteterne placered omkring transition-fasen. I denne fase forekommer der nogle aktiviteter. De typiske aktiviteter, er: Beta- eller accepttest for at kunne validere de krav, produktet skal opfylde. Samtidig med dette er håndteringen af feed-backen vigtig. Har man styr på dette, kan man hurtigt se, hvilke bugs der skal laves, hvor man efterfølgende hurtigt kan udgive den nye rettede version af produktet. En anden aktivitet kunne være installationen af en given release for kunden. Har kunden et ældre system, kan det omfatte aktiviteter for integration af det nye system. Ud over det skal administratorer/brugere undervises i, hvordan tingene hænger sammen, og hvordan man har tænkt. Efter dette kan man færdiggøre eventuelle projektartefakter, hvor man efterfølgende kan begynde at tage en beslutning om, hvornår projektet menes at være afsluttet.



Ser man i stedet på den agile udvikling og deployment, er et punkt inkrementel levering af forskellige releases. Som tidligere nævnt i rapporten User stories en vigtig del af den i den agile måde at arbejde på. Og for løbende at kunne afgøre om en User story er færdig, skal der laves en accepttest på den. Dette kan gøres ved at lave automatiske accept- og unittests. Nogle mener, at et deployment-sprint, i den agile verden, er overflødigt, hvis teamet overholder og følger de agile idealer. Og da ikke altid er helt muligt, er det godt at have med. Der anvendes derfor, i agile sammenhæng, accepttests for både sikre sig, at hver enkelt User story er helt ’Done’, og for lave en afsluttende test af produktet, lige inden det frigives. Det med at User storien skal være done, er, at koden er tjekket af teamet og sat i verify, og accepttesten er gennemført. Det eneste problem der kan være ved disse accepttests, er, at det kan være svært at estimere og indsætte accepttestene i sprintende, da man under hele forløbet tester og debugger produktet, indtil man til sidst har en version, der menes at være god nok til en release. Det vil sige, at man kan have nogle ikke-done User stories før indtil helt til sidst inden releasen. Så derfor kan det netop være godt, at teamet afsætter noget tid på rettelser i det efterfølgende sprint.

# Konfigurationsstyring:

Konfigurationsstyring hvad er det? Det er, hvis vi antager, at der foretages ændringer i et system, så skal der etableres en ny version af systemet. Det kan f.eks. være at systemet skal kunne køre på flere forskellige operativsystemer, eller ved at systemet har forskellig funktionalitet alt efter, hvad operativsystem, det kører på. Ud over det kan det være, at man har været nødt til at skræddersy nogle specifikke krav for forskellige kunder. Man kan derfor snakke om, at det er en del af den kvalitetssikring, man bør have, at systemændringerne styres. En af grundende til at konfigurationsstyring er vigtig, er fordi man som programmør retter en eventuel fejl, hvor man så efterfølgende opdager, at dette ’fix’ har medført nye og stører problemer i koden. Har man ikke styring over de ting, der er blevet rettet, kan man til sidst ende med ikke at kunne finde den oprindelige version af koden. Er dette tilfældet, kan det koste programmøren lang tid at skulle rekonstruere koden. Ikke nok med det koster en masse dyrebar tid, så kan det efterfølgende føre til yderligere fejl i den nye kode. En anden god grund til at anvende konfigurationsstyring er i forhold til kundeservicen. Forestil dig at en kunde ringer ind og stiller følgende spørgsmål: ”Hvilken udgave af softwaren, kunden har”, ”Er det givne problem kendt i forvejen, og har andre klaget over det samme” og ”Om der er et fix på vej og måske et release på dette”. Har man konfigurationsstyring på det, er det let at gå ind og se oplysningerne på disse spørgsmål, som en eventuel kunde kunne komme med.

## Change management:

Change management vedrører ændringsstyring, og er et værktøj, der bruges af et udviklerteam, til styring og kontrol af fremdriften i et givent ændringsønske. Ved at anvende dette gøres det lettere for programmørerne at få et bedre overblik over eksempelvis designændringer. Samtidig med det sikrer man sig, at alle ændringer der skal foretages, bliver håndteret efter aftalte standarder, så kvalitetsniveauet forbliver højt. Igen skal man sikre sig, at eventuelle fordele og ulemper ved en kundes ændringsønske bliver vurderet ordenligt, for man derefter kontrolleret kan foretage ændringer med minimale omkostninger.

Snakker man om change management i den agile verden, ses det ofte, at kontakten mellem kunden sker gennem en productowner, der efterfølgende giver teamet besked om eventuelle ønskede ændringer. I andre tilfælde er der direkte kontakt med kunden. Sker kontakten via en productowner, vurderes effekten af ændringen i samarbejde med teamet. På denne måde kan prioriteringer foretages i forbindelse med planlægning af den kommende iteration. Efterfølgende foreslås forbedringer af systemets kvalitet af teamet, hvorefter der bliver lavet noget refactoring af koden. Det kan næsten ikke undgås at lave refactoring i en sådan udviklingsproces, så derfor anses denne proces ikke for at være en ekstra omkostning.

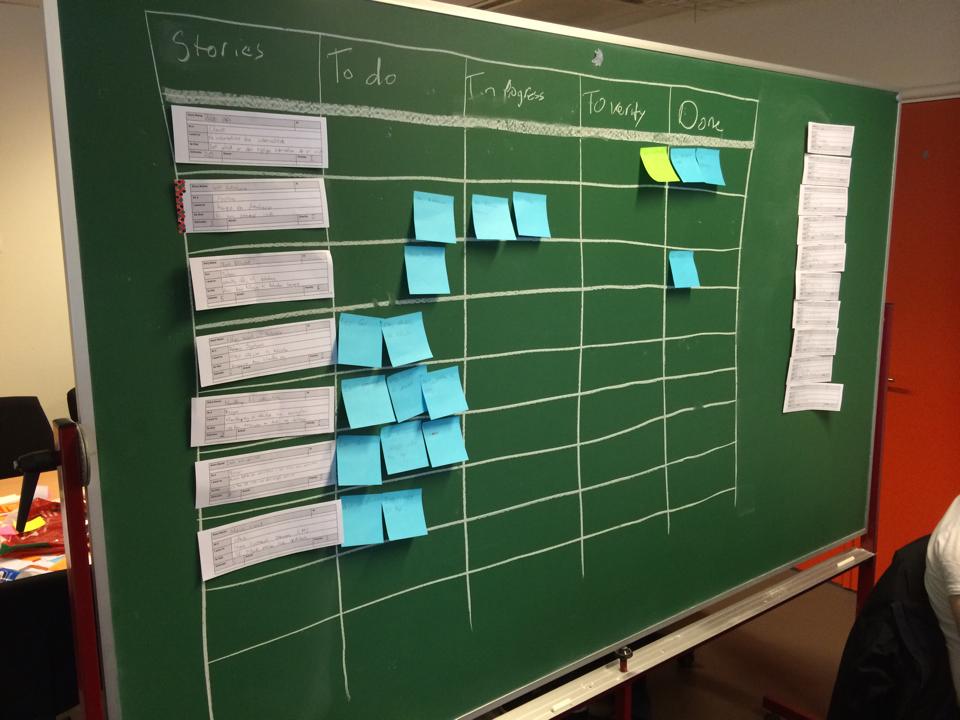
# Virksomhedsbesøg:

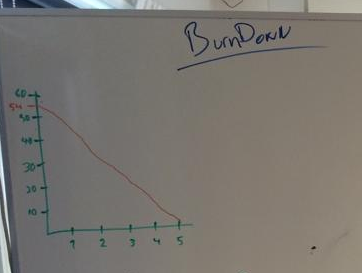
Vores gruppe blev delt op i to. Den ene gruppe besøgte virksomheden EG A/S, hvor den anden gruppe besøgte ???. Hos EG A/S udvikler de primært i C# og .Net. Ud over dette kan de også udvikle i Dynamics AX, X++ samt HTML5/JavaScript. De produkter EG udbyder, er: Project and Process oriented Manufactoring (Fremstilling), Manufactoring execution (Produktion, maskiner og hurtigere setup), Time & Experience, including mobile (Tidsregistrering, løn samt kørsel) og til sidst Utility Billing & Grid maintenance (Zynergi og databehandling).

# Sprints:

## Sprint 1:

Formålet med et sprint, er, efter man har valgt længen pr. sprint (6 dage i vores tilfælde), at man kan planlægge rimelig præcist, hvor meget man kan nå at lave i et sådan sprint. Måden det gøres på, er ved at man laver en tabel med de forskellige User stories og tasks som vist på billedet nedenfor. Så er det meningen, at man hver morgen, som det første, starter med at snakke om, hvilke tasks der skal laves i løbet af dagen, og hvem der laver hvad. Efter dette kan folk fortælle, hvad de nåede at lave dagen forinden. Samtidig med denne tabel, bruger man en burndown-model til at sørge for at overholde planen for, hvor mange point det er muligt at lave i et sprint. Til højre ses vores udgave af en burndown-model for sprint 1. Idet vi var fem i gruppen dannede vi tre par: to tomandspar og én, der arbejdede alene. Da vi havde fire timers arbejde pr. dag pr. par, regnede vi os frem til, at vi kunne nå 54 timer på et fem dages-sprint. På tabellen nedenfor ses de stories og tasks, vi havde med i sprint 1. Disse tasks estimerede vi ved hjælp af Planning Poker. Vi estimerede hver tasks i timer, hvorefter vi uddelte tasksene til hvert par. Første dag





|  |  |  |  |
| --- | --- | --- | --- |
| Story | Tasks |  |  |
| Web-Api | Lave metode der returnerer true eller false | Domænemodel | Lav modellag |
| Opret database | Opret database diagram | Opret tabels | Lav database |
| Opret aktivitet | Lav Gui | Tilføj kald til Api |  |
| Tilføj event til kalender | Lav Gui | Tilføj afholder | Tilføj kald til Api |
| Info om aktiviteter | Lav Gui | Lav controller | Metode til Api |
| Admin klient | Lav Gui | Tilføj kald til Api |  |
| Tilmelding til aktivitet | Tilføj bruger til aktivitet | Lav pop-up | Opret metode til Api |

var Simon og Marck et par, og de lavede Gui til ’opret aktivitet’. Det var meningen, de skulle have oprettet databasen med tilhørende tabels også, men pga. af problemer med Sql Server Management Studio nåede de det ikke. Mathias var alene den dag, og lavede en prototype af applikationen – hovedsagelig menu-delen. Nicolai og Kim fik lavet forbindelsen til web-Api ‘en, modellaget dertil samt en domænemodel. Anden dag i sprintet oprettede Marck og Simon databasen med tabels. Nicolai og Kim lavede modelklasser og Mathias sad fortsat og arbejdede på prototypen. Den tredje dag var både Kim og Mathias syge. Mathias + Kim = syg. Da vi kun var 3 fremmødte, blev vi nødt til at droppe pair-programming fra morgenstunden af. Vi brugte sund fornuft. Marck rettede lidt i ‘Opret aktivitet’ og startede på ‘Tilføj til kalender’. Simon & Nicolai lavede test first og hvordan man tilføjer data til databasen. Fjerde dag i sprint 1 var Kim stadig syg. Der brugte vi hurtigt 10-15 minutter på review af koden, så alle havde fælles kodeejerskab. Mathias og Marck lavede ’Opret Aktivitet’ med check-boxes. Simon og Nicolai lavede databaseklasser for ’Opret aktivitet’, ’Location’ samt ’Event’. Femte og sidste dag lavede Marck & Mathias lavede Gui til Event, CreateActivity & CreateLocation færdigt. Ud over dette fik de programmet til at tjekke op på, om alle felter er udfyldt, før de kan blive oprettet. Simon, Nicolai & Kim prøvede noget nyt. Da vi var fuldtallige og ingen af os skulle være alene, sad de tre sammen. De fik fikset DB-klasser og påbegyndte CTR-klassserne. LocationCtr & ActivityCtr virker - EventCtr drillede lidt. Hertil blev der lavet test-first.