# Idégenerering:

# 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.

# Projektplanlægning:

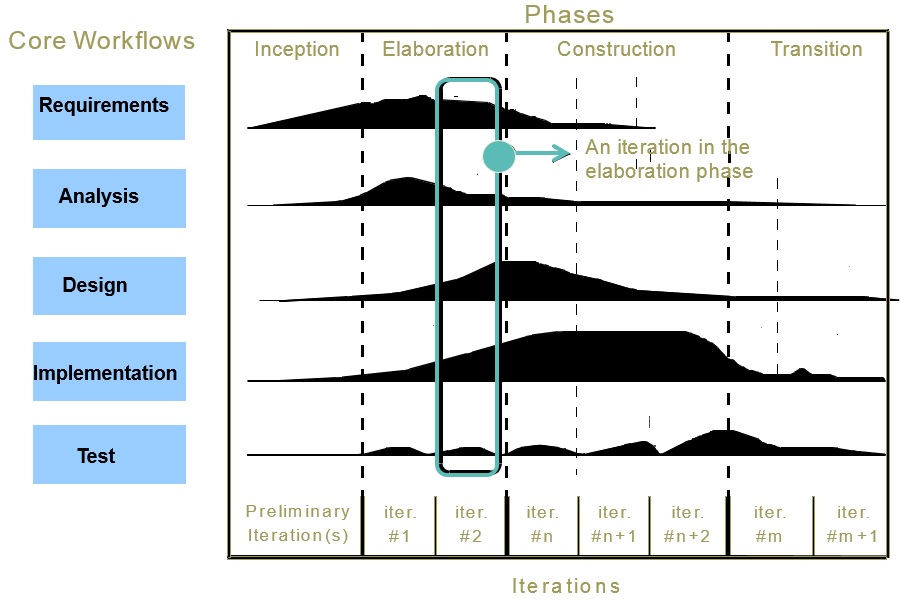
I dette afsnit vil der bliver beskrevet nogle teknikker til planlægning af et projekt.

## Generelle teknikker:

Nogle af de generelle teknikker der vil blive beskrevet herunder, kan indgå i både Plandrevene projekter og i Agile projekter.

## Projektmodel:

En projektmodel er en model bestående af faser og milepæle. Disse angiver hvad der skal arbejdes med i de forskellige iterationer i projektet. En fase er en bestemt periode i projektet hvor der arbejdes med de angivende opgaver som fremgår af modellen, hvilke opgaver der arbejdes med bestemmes af milepælene. En milepæl er det resultat som en fase skal ende ud med, det kan f.eks. være når en prototype er færdig, eller når en bestemt del af projektet er blevet testet.



Figur 1 Projektmodel

## Projektplaner:

En projektplan er en aftale om et projekts forløb, så både kunden og projektteamet kan se hvilke aftaler der er blevet indgået. En projektplan laves med udgangspunkt i projektmodellen som blevet forklaret tidligere. Projektplanen består af en ekstern plan og en intern plan. Den eksterne plan er aftalen mellem projektteamet og kunden, den omhandler primært tid, ressourcer og leverancer. Den eksterne plan gælder for hele projektet. Den interne plan er en aftale der gælder for projektteamet, den beskriver hvordan man skal løse de opgaver som den eksterne plan omhandler. Den interne plan glæder for de enkelte faser / iterationer.

## ”Work breakdown”:

For at gøre et større projekt mere overskueligt at planlægge, bryder man ofte dele / features af et projekt op i mindre tasks. Grunden til dette er at det gør det nemmere at uddelegere opgaver til de enkelte projektdeltagere, samt tildele ansvarsområder for de enkelte tasks. Det bruges også for nemmere at kunne estimere hvor lang tid der skal bruges for at få enkelte tasks færdige.

Der kan selvfølgelig være visse risici forbundet med at dele features op i forskellige tasks, derfor er det ofte vigtigt at der indgår erfarende udviklere i denne proces, eller at man evt. inddrager en ekspert på området til at hjælpe med processen.

Tasks må ikke være større end at de kan laves på en enkelt arbejdsdag. Hvis det ikke er muligt at fuldføre en task inden for en arbejdsdag, skal den opdeles i yderligere tasks.

## Estimering af personressourcer:

For at kunne lave korrekte estimeringer, er det vigtigt at man får lavet en realistisk plan angående personressourcer. Derved menes at man skal have vurderet hvor mange mandetimer man har til rådighed i et projektteam, i løbet af en arbejdsdag. Formålet med dette, er at man så kan planlægge projektet ud fra hvor mange timer de forskellige opgaver estimeres til.

Et godt estimat bygger på erfaring, faglig viden og god argumentation for det estimat man kommer frem til. Estimering vil også bliver nemmere jo flere gange den enkelte udvikler har været igennem estimeringsprocessen, samt hvorvidt der er tale om en type opgaver man er bekendt med.

## Estimeringsmetoder:

Der findes fire forskellige estimeringsmetoder, Analogi, Nedbrydning og sammentælling, Faktorvurdering og Planning Poker.

Analogi går ud på at man sammenligner med gamle projekter, hvor man tager højde for mindre forskelligheder. Det giver det bedste resultat hvis man tidligere har arbejdet med lignende projekter. Det kræver dog at der er opsamlet data fra de gamle projekter, hvilket ikke altid er tilfældet.

Nedbrydning og sammentælling er en metode hvor man opdeler opgaver som virker uoverskuelige, til mindre dele, indtil opgaverne kommer ned i en størrelsesorden hvor man føler det er til at sætte et timetal på. For at komme frem til det endelige estimat for et helt projekt når man anvender denne metode kan man anvende ”Successiv kalkulation”, det er en metode der bruges til at vurdere et overslag af en given opgave, den kan anvendes til forbrug af f.eks. tid eller penge, alt efter hvad man ønsker et estimat på.



Figur 2Successiv kalkulations skema

Ovenstående skema viser et eksempel på Successiv kalkulation, derud fra kan se at der er blevet estimeret et minimum forbrug, et sandsynlig forbrug og et maksimum forbrug. Ud fra disse estimater udregnes en middelværdi, en standard afvigelse og en varians, ud fra de udregninger det ses under skemaet. Når alle estimater er udregnet ligges alle middelværdier samme plus standard afvigelsen for hele projektet, og derved opnår man så det endelige estimat inc. afvigelse, derved opnår man et estimat der kan antages at være realistisk.

Faktorvurdering er en metode hvor man anvender en algoritme til at estimere omfanget at et projekt.

Planning Poker er den sidste estimeringsmetode, her er estimeringen story baseret. En story bliver taget op af gangen, hver medlem af projektteamet har et sæt kort som har følgende relative værdier: 0, 1, 2, 3, 5, 8, 13, 20, 40 og 100. Hvert medlem vælger så et kort, ud fra den story der skal estimeres, og lægger det på bordet samtidig med de andre medlemmer. Der skal så argumenteres for det valgte estimat, her er det vigtigt at have fokus på det højeste og laveste estimat, da disse kan afspejle en misforståelse af den givende story, eller påvise at storien er mere omfattende end antaget af de andre medlemmer. Efter der er argumenteret tager man så en nye runde planning poker, og dette fortsætter til der er enighed.

## Projektovervågning og styring:

Overvågning af projektplanen kan foregå på forskellige måder. Det kan gøres ved at følge op på aftaler med f.eks. samarbejdspartnere og/eller leverandører, så man er sikker på at disse aftaler overholdes, så det ikke få negativ indflydelse på projektet. Man kan foretage risici analyser undervejs, så man hele tiden har styr på hvordan man skal reagere hvis der opstår et problem som man har lavet en risikoanalyse over. Man kan holde øje med bemandingen på projektet, hvor man så kigger på om de personressourcer og kompetencer man har planlagt ud fra også stemmer overens igennem projektet. Man kan kigge på motivationen hos medarbejderne, samt deres ansvarsområder. Yderligere kan man holde øje med om der kommunikeres tilstrækkeligt imellem medarbejderne.

For at få et bedre overblik undervejs i projektet er det også en god ide at lave proces reviews, hvor der kigges på de produkter der bliver lavet undervejs. Det samme gælder milepæls reviews, hvor man ser på om man er kommet i mål i forhold til de kriterier der er opstillet for den milepæl man har gennemført.

Udførelse af korrigerende handlinger sker ved at man indhenter en accept hos kunden før man lave en ændring i planen, først herefter kan planen så ændres.

## Plandreven planlægning:

Et projekt der laves ud fra plandreven planlægning vil typisk gøre brug af vandfaldsmodellen. Planen er baseret på at man har flere udviklingsfaser, der hver især skaber et grundlag for den videre udvikling. Planlægningen sker først i projektperioden og udføres af projektledelsen, aktiviteter og leverancer låses fast i en tidsplan, så projektteamet og kunden hele tiden kan have et overblik hvad der skal laves hvornår, og hvornår der kan forventes noget som kan vises og afprøves af kunden.

## Projekt planer:

En projektplan skal indeholde følgende:

* En introduktion til projektet så man kan se hvad projektet går ud på.
* Projektets organisering, så man kan se ud fra planen hvem der skal arbejde med hvad, og hvem der har ansvaret for at kravene bliver overholdt.
* En risiko analyse, så det fremgår hvilke udfordringer man skal være særlig opmærksom på, samt hvordan man skal reagere hvis et problem opstår.
* Krav til hardware og software, så man er sikker på at alle i projektteamet har samme version af udviklingsværktøjer, og at deres hardware passer til projektet.
* Work breakdown, så man har et overblik over hvordan større dele af projektet skal blive udviklet, på en mere overskuelig måde.
* Tidsplan, så man hele tiden kan se om man er på rette kurs i projektet, om det hele følge planen, eller om man er nødt til at skulle foretage justeringer.
* Forhold vedr. overvågning og afrapportering.

## Risikoanalyse:

En risiko skal ses som en mulig negativ planafvigelse. Risiko analysen laves for at give et overblik over hvilke udfordringer man kan løbe ind undervejs i projektet, og skal bruges til at kunne vise hvor sandsynlig de forskellige risici er, samt hvordan de skal behandles hvis problemet opstår.

Trin i risiko analysen:

* Identificer risici (teknologi, udviklere, organisation, værktøjer, krav og estimater)
* Tilordn sandsynlighed
* Vurder konsekvenser
* Overvej håndtering

Det er vigtigt at have fokus på de største risici.

## Ansvarsforhold:

Projektlederen har ansvaret for at lave projektplanen ved opstart af et nyt projekt. Projektlederen skal lave en overordnet plan for hele projektet, samt lave interne detailplaner for de enkelte afdelinger/teams.

Projektlederen har også ansvaret for løbende at følge op på status og milepæle for projektet, så han kan fortage justeringer hvis det er nødvendigt. Krav, tid og pris er ofte låst fast i et plandreven projekt, så disse parametre kan der ikke ændres på.

Udviklerene har ansvar for at udfører arbejdet ud fra de specifikationer der fremgår af planen. Hvis de løber ind i udfordringer, eller kan se at noget ikke kan lade sig gøre i forhold til planen, skal denne kommunikation ske igennem projektlederen.

## Agile planlægning:

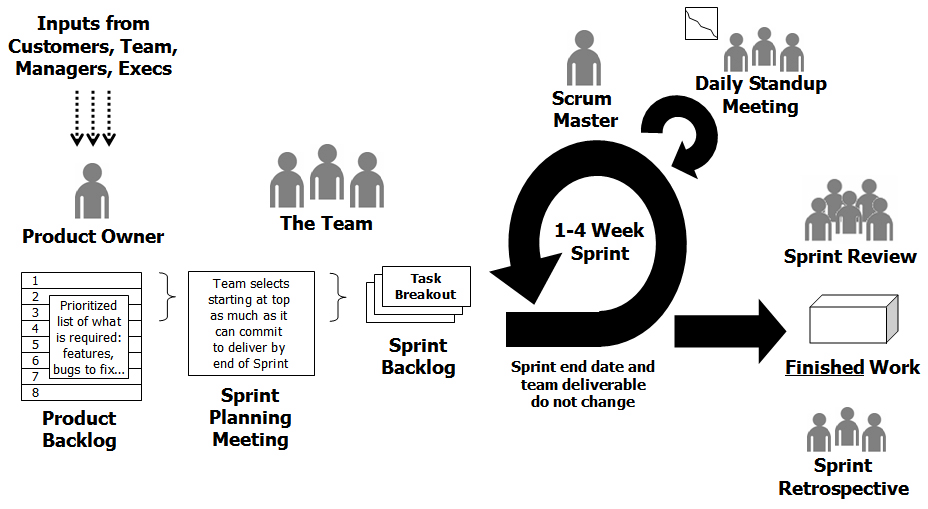
Specifikation, design, implementering og test er vævet ind i hinanden og output fra udviklingsprocessen fastlægges gennem forhandlinger gennem hele udviklingsprocessen

Agile planlægning er tilpassende i forhold til plandreven planlægning, forstået på den måde at i agile planlægning har man hele tiden kontakten med kunden / repræsentanterne. Man arbejder med iterative udvikling og tilpasser planen ud fra det feed back man får fra kunden efter et sprint er færdig.

I agile planlægning arbejdes der med selvstyrende teams. Teamet består af professionelle udviklere som alle laver alting i projektet. Teamet tager beslutningerne omkring planen for udviklingen samt hvilken teknologi der skal anvendes. I agile planlægning bruges der ikke tid på dokumentation, alt kommunikation sker face to face på scrum meetings og sprint reviews.

## Planlægningsprocessen i Scrum:

Planlægningsprocessen foregår løbende når man arbejder med scrum, nedenstående figur illustrere hvordan processen foregår.

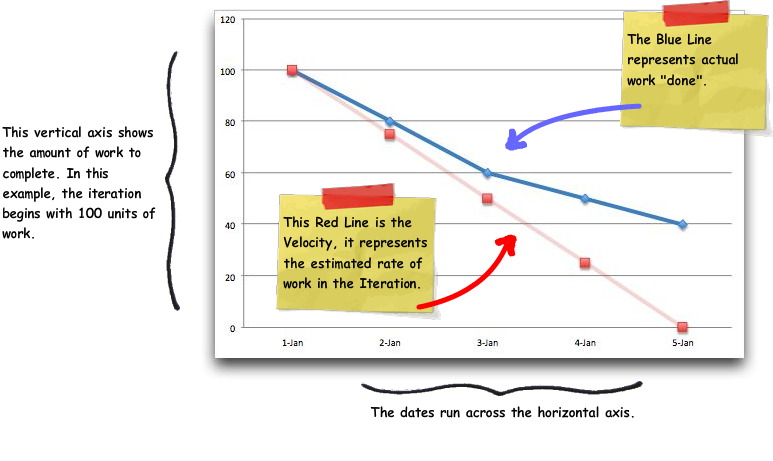


Figur 3Planlægning i Scrum

Product owner er den person der har kontakten til kunden, og har aftalt med kunden hvad projektet skal indeholde. Scrum teamet afholder så et sprint planning meeting, hvor det bestemmes hvad der skal med i det kommende sprit, man har så en sprint backlog hvor man kan hente tasts ind fra hvis man skulle blive færdig med nogle opgaver hurtigere end estimeret i løbet af sprintet. Scrum master og teamet holder hver dag standup meeting, hvor dagens opgaver uddelegeres. Efter hvert sprint afholdes et sprint review hvor man kigger på hvordan sprintet er gået, til dette kan kunden være tilstedet og man får feedback på det arbejde der er lavet. Dette tages så op til sprint retrospektive og man tilpasser derefter sin plan for næste sprint.

## Burndown:

Burndown er en vigtig del af planlægningen i et Scrum projekt, da det er her man dag til dag kan se om man følger planen, og om de estimater man er kommet frem til nu også passede.



Figur 4Burn Down eksempel

Figuren viser et eksempel på en burndown, hvor man ikke helt har nået de opgaver man havde estimeret at man kunne når inden for sprintet. Måden man kan håndtere dette på i Scrum er ved at tage en eller flere tasks ud af sprintet, så man stadig kan nå i mål, disse tasks kommer så tilbage i sprint backlog’en.

## Ansvarsforhold ved Scrum:

I Scrum arbejder man som tidligere nævnt med selvstyrende grupper, hvilket også gør at ansvaret for sprint planlægningen, opfølgning via standup møder, samt kvalitetssikring af udviklingsarbejdet ligger hos Scrum teamet. Teamet er også ansvarlig for korrektion ved afvigelser.

Produkt owner har ansvaret for at releaseplanen overholdes, for at produkt backlog’en er opdateret, og er ansvarlig for godkendelse af et sprint hos kunden.

## Styringsparametre:

Tid og penge er låst fast i Scrum, så det eneste parameter man kan skrue på kravene eller scope som det også kaldes.

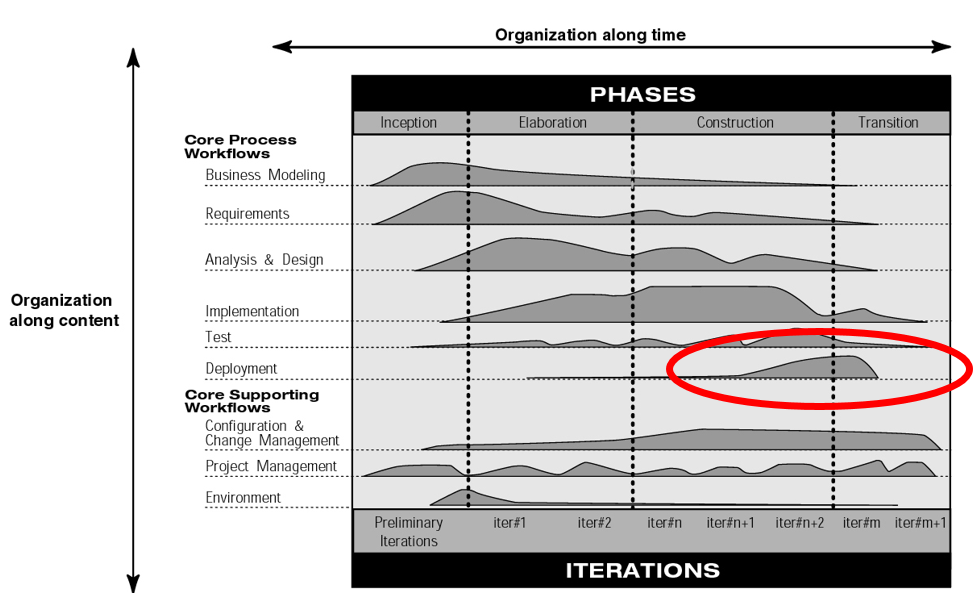
# Deployment:

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:

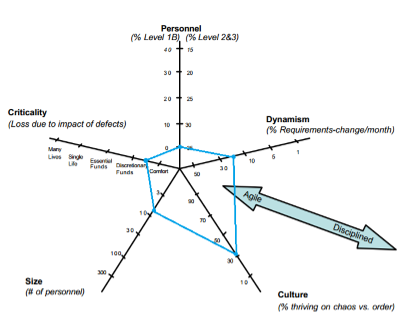
Change management vedrører ændringsstyring, og er et værktøj, der bruges af et udviklerteam, til styring og kontrol af fremdriften i et givent ændringsønske. Ved at anvende dette gøres det lettere for programmørerne at få et bedre overblik over eksempelvis designændringer. Samtidig med det sikrer man 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:

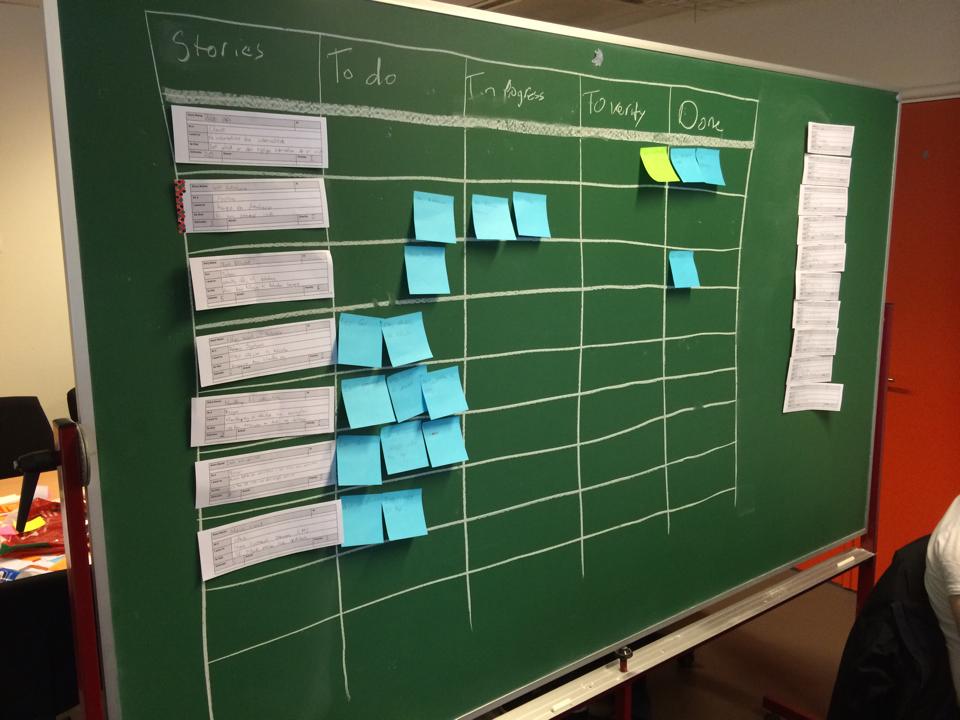
Vores gruppe blev delt op i to. Den ene gruppe besøgte virksomheden EG A/S, hvor den anden gruppe besøgte ???. Hos EG A/S udvikler de primært i C# og .Net. Ud over dette kan de også udvikle i Dynamics AX, X++ samt HTML5/JavaScript. De produkter EG udbyder, er: Project and Process oriented Manufactoring (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). Måden EG styrede deres projekter på, var ved hjælp af SCRUM. Længden på deres sprints er tre uger, og til hver release skal de bruge fire sprints. Det vil sige, at de på tre måneder har tre udviklingssprints og et stabilitetssprint. Ud over dette har de en fast kodestandard, der gør, at alle teams følger den samme cyklus, og medlemmer kan derfor let flyttes rundt. Det giver en form for tryghed, og overholdes en deadline ikke, så går det ud over hele organisationen. Deadlinen skal overholdes, for den kan ikke flyttes på. Deres core-team består af en Product owner (en medarbejder fra EG’s salgsafdeling), to udviklere og en tester. Ved større teams bliver en tester samt en af udviklerne gjort til lead. På den måde sikres det, at standarterne overholdes, og at kvaliteten forbliver høj. Ham navn!! er Scrum master for alle teams, samtidig med at han er chef for hele afdelingen, hvilket kan være en svær opgave. Ligesom os brugte de burndown og samme estimeringstid, hvilket var på en time. Deres burndown var dog noget pænere end vores. De bruger dog ikke Planning Poker til at estimere deres opgaver, men deres sunde fornuft til internt i teamet at estimere hver Story. De point de kan give til hver Story, er hhv.: 2, 4, 8, 16, 32, 64, 128 og 256. Kommer opgaven op 128 og 256, tages den op til en alvorlig diskussion. Måden de tester koden på, er ved hver nat at lave et build af systemet, for at der efterfølgende bliver startet en automatisk test. De fejl, der bliver fundet, indrapporteres, og en tester tager direkte kontakt til den ansvarlige udvikler. Hver feature bliver funktionstestet så tideligt, det kan lade sig gøre. Til sidst når de er nået til stabilitetssprintet, laver de End-to-End-tests.

De har nogle forskellige måder at lave reviews på. Et af dem er reviewet efter hvert sprint. Her ser de på, om der skal ændres på featurelisten, og om sprintet har været fyldestgørende. I deres retrospektive udveksler de hinandens erfaringer, og hvad de skal tage med og passe på. Til sidst har de et Management review, hvor Scrum masteren deltager sammen med andre ledere. Dette review afholdes lige inden en release. Vi har prøvet at sætte dem ind i Boehm’s, som kan ses til højre. (mere forklaring) og måske noget om backlog



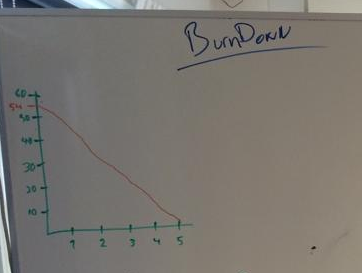
# Sprints:

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.



## Sprint 1:

Vores primære mål for dette sprint, var at få Web-Api‘en til at virke. Det var det på grund af, atTil 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 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 på prototypen af applikationen – hovedsagelig menu-delen. Nicolai og Kim fik



|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

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 tre 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.

I sprintet lavede vi så vidt muligt Pair Programming hver dag. Den eneste dag det fungerede helt optimalt, var fjerde dag i sprint 1. Kim var den eneste, der var syg den dag, så vi kunne derfor dele os op i to par. På den måde var der ingen, der skulle sidde alene, og vi undgik at lave Pair Programming tre sammen. Som nævnt ovenstående prøvede vi det dog den femte dag i sprint 1, og det gik udmærket, men stadig ikke helt optimalt.

Stories bedre----

## Sprint 3:

I det sidste sprint var vi nødt til at finde ud af, hvad vi skulle gøre for at nå at blive færdige til tiden, eftersom vores burndown fra sprint 2 næsten havde flatlined. Vores burndown fra sprint 3, kan ses her. Da vi ikke nåede nogle af vores mål, lod vi tre af Stories’ne hænge og ændrede på den sidste. Vi regnede med kunne nå de sidste 46 point på de sidste fem dage. Der var ingen fraværende i dette sprint, så vi kunne derfor holde standup meeting hver dag. Efter dette lavede vi lidt review for at alle fik fælles kodeejerskab Idet vi med lavede Pair Programming, var det Nicolai, der skulle sidde alene for at løse problemet fra sprint 2. Men vi var desværre rendt ind i et større problem, end vi havde regnet med. Det resulterede i, at vi sad mere eller mindre fast et par dage. På tredje dag i sprintet blev problemet løst, men vi havde desværre måtte bruge for lang tid på det, så vi var langt bagud i forhold til vores burndown. Det endte med at vi ikke blev færdig med produktet, hvilket ikke havde været godt i den virkelige verden. Vi kom frem til, at vi havde fejlestimeret, prioriteret og lavet dårlige User stories, og at det var de største grunde til, vi ikke nåede det inden for deadlinen. Samtidig med det var vi ikke helt sat ind, hvordan man skulle lave dette (forklaring), så det skal vi nok være bedre til næste gang.

