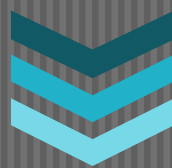


# LevelUp



## Semester 4, gruppe 3

Denne rapport er udarbejdet som en del af Systemudvikling på 4. semester på Datamatikeruddannelsen på UCN. Projektet bundler i brug af Extreme Programming til udvikling af en software applikation, som skal motivere brugeren til at motionere. Projektet er udarbejdet gennem perioden 25/11-2013 til den 7/1 -2014.

R o n n i e   H e m m i n g s e n

R a s m u s   M e y e r

C h r i s t o f f e r   F r e d e

T o k e   O l s e n

## 1 Indholdsfortegnelse

2	Teori.....	5
2.1	Projektplanlægning .....	5
2.2	Unified Process.....	6
2.2.1	Inception .....	6
2.2.2	Elaboration.....	6
2.2.3	Construction.....	6
2.2.4	Transition .....	6
2.2.5	Hvordan er UP anderledes fra den Agile Systemudviklingsmetode ? .....	6
2.2.6	UP.....	6
2.3	Agilt .....	7
2.4	Scrum.....	8
2.4.1	Scrum artefakter .....	8
2.4.2	Scrum roller – Product Owner .....	9
2.4.3	Scrum roller - Scrum Master .....	10
2.4.4	Scrum roller – Development Team .....	10
2.4.5	Scrum Events.....	10
2.4.6	Scrum Events – Sprint Planning .....	11
2.4.7	Scrum Events – Daily Scrum.....	12
2.4.8	Scrum Events – Sprint Review.....	13
2.4.9	Scrum Events – Sprint Retrospective .....	13
2.5	Extreme Programming .....	15
2.5.1	12 arbejdspraksisser .....	15
2.5.2	Planning Game .....	15
2.5.3	Små udgivelser med korte mellemrum .....	15
2.5.4	Systemmetafor.....	15
2.5.5	Simpelt design.....	15
2.5.6	Test.....	16
2.5.7	Hyppig refaktorering.....	16
2.5.8	Parprogrammering.....	16
2.5.9	Fælles ejerskab af programkode.....	16
2.5.10	Kontinuerlig integration.....	16

2.5.11	Overkommeligt arbejdstempo.....	16
2.5.12	Et samlet udviklingshold .....	16
2.5.13	Fælles kodestandard .....	16
2.6	Kanban.....	17
2.6.1	Scrum + Kanban .....	18
2.7	Arkitektur .....	19
2.8	Metodevalg .....	20
2.9	Risikostyring .....	21
2.9.1	Risikostyring i Scrum .....	21
2.9.2	Kvalitetssikring .....	21
2.10	Krav Specificering og Vedligeholdelsesstyring .....	25
2.11	Deployment og Configuration Management .....	27
2.11.1	Deployment.....	27
2.11.2	Deployment fase .....	27
2.11.3	Configuration management.....	28
2.12	Product Vision .....	29
2.12.1	Navn og Applikation Statement .....	29
2.12.2	Behov .....	29
2.12.3	Produkt.....	29
2.12.4	Værdi.....	30
2.12.5	Konkurrence.....	30
2.12.6	Målgruppen.....	30
2.12.7	Brugsscenarier .....	31
2.12.8	Login.....	31
2.12.9	Log træning: .....	32
2.12.10	Ved level up.....	32
2.12.11	Profiloversigt: .....	32
2.12.12	Leaderboards:.....	32
2.12.13	Challenges: .....	32
2.12.14	Statistik:.....	32
2.13	Kanvas .....	33
2.13.1	Værdiskabelse .....	33

2.13.2	Kundesegment .....	33
2.13.3	Nøglepartnere .....	33
2.13.4	Primære resourcer .....	34
2.13.5	Kunderelationer .....	34
2.13.6	Kanaler .....	34
2.13.7	Omkostninger .....	34
2.13.8	Indtjeningsmuligheder .....	34
3	Sprint 0.....	35
3.1	Retrospective .....	35
3.2	Metodevalg .....	36
3.2.1	Criticality .....	36
3.2.2	Culture.....	36
3.2.3	Dynamism .....	37
3.2.4	Technology.....	37
3.2.5	Personnel .....	37
3.2.6	Size .....	37
3.2.7	Valget .....	38
3.2.8	Risikostyring .....	39
3.2.9	Risikoanalyse for LevelUp .....	40
3.2.10	Arkitektur .....	41
4	Sprint 1.....	42
4.1	Planlægning .....	42
4.1.1	Velocity .....	42
4.1.2	Planning poker .....	42
4.1.3	Fra productbacklog til sprintbacklog .....	42
4.2	Review .....	43
4.3	Retrospective .....	43
4.3.1	Hvad var godt ? .....	43
4.3.2	Hvad var ikke så godt ? .....	44
4.3.3	Hvad kunne vi gøre bedre til næste sprint?.....	44
5	Sprint 2.....	45
5.1	Introduktion .....	45

---

5.2	Planlægning .....	45
5.3	Retrospective .....	46
6	Sprint 3.....	49
6.1	Planlægning .....	49
6.2	Review .....	49
6.3	Retrospective .....	50
7	Overordnet retrospektive for hele forløbet (Perspektivering).....	52
7.1	Hvad gik godt?.....	52
7.2	Hvad gik ikke så godt? .....	52
7.3	Generelle forbedringer til hele forløbet .....	52
8	Konklusion .....	53
9	Kildeliste .....	54

## 2 Teori

### 2.1 Projektplanlægning

Inden den reelle programmering begynder er det nødvendigt at lægge kræfter i planlægningen af forløbet. Der skal vælges strategi, risiciene skal udforskes og overvejes, metode skal vurderes og vælges og værktøjer skal tages i brug.

Før alt andet skal det være klart, hvad formålet med projektet er, og hvilke mål der skal opnåes. Målene skal danne styringsgrundlaget, og bør overholde følgende krav:

- Målene skal være målbare, dvs. det skal være muligt præcist at vide, om målet er nået eller ej. Målet kan eksempelvis være funktionelt, hvor et system skal kunne håndtere et vis antal brugere på samme tid. Der kan dog være svært at stille præcise målbare krav til samtlige mål. Eksempelvis kan et mål omhandle en ikke-målbart faktor som brugervenlighed, men så skal der anvendes undersøgelser mv. for at gøre det målbart.
- Målene skal bruges til at udtrykke de resultater, som man ønsker opnået fremfor de midler, som man ønsker at opnå resultatet med.
- Målene skal ligge indenfor projektets handlerum, dvs. begrænse målene til kun at løse den opgave, som er blevet stillet.

Når dette er på plads skal der vælges udviklingsstrategi, hvor der oftest tales om plandreven udvikling og agil udvikling. Hvilken af disse der vælges afhænger i høj grad af, hvilken type opgave, der er stillet.

I den plandrevne udviklingsprocess planlægges detaljerne fra starten, hvilket er nyttigt, når processen og løsningen er velkendt. Denne udviklingsproces kan dog være problematisk, hvis ikke forudsætningerne for projektet holder stik igennem forløbet.

I modsætning til den plandrevne proces er den agile proces meget fleksibel, da planlægningen er meget løs og detaljerne først fastsættes under selve udviklingen. Denne proces bruges til projekter, som har mange ukendte faktorer, eksempelvis ny teknologi, der skal tages i brug. Agil udvikling kan dog blive problematisk ved større projekter, da den ikke tager højde for fremtidig funktionalitet under forløbet. Dette kan betyde spildt arbejde, hvis den eksisterende programkode konstant skal tilpasses for at støtte den kommende kode. Samtidig er agil udvikling ikke velegnet til projekter, hvor der er store krav til kvaliteten, da der ikke tages højde for de overordnede sikkerhedshuller, når den enkelte funktion bygges.

## 2.2 Unified Process

I dette afsnit vil vi forklare hvad UP procesmodellen er og hvordan man bruger den. På et overordnet plan kan det siges at, modellen bruges til at organisere aktiviteter og artefakter som leder til produktion og udvikling af et færdigt it-system. Dette gøres for at vise at vi kender teorien bag det og at vi ved hvordan man bruger det.

UP som er en forkortelse af Unified Process er en iterativ Systemudviklingsmetode. UP procesmodellen består overordnet set af 4 faser – Inception, Elaboration, Construction og Transition.

### 2.2.1 Inception

Inception er ikke en kravspecifikation som det er i vandfaldsmodellen. Det er i stedet en kort fase hvor man indsamler nok information til at man kan træffe en beslutning om projektet skal fortsætte eller ej. En kandidatarkitektur identificeres og der udarbejdes design af systemets nøglefunktioner. Disse nøglefunktioner beskrives i form af use cases.

### 2.2.2 Elaboration

I Elaboration fastlægges den identificerede kandidatarkitektur og der udarbejdes en prototype som skal fungere som arkitekturens grundfundament. Systemets risici vurderes og der udarbejdes prototyper for at eliminere dem. Resultatet er at have en eksekverbar arkitektur.

### 2.2.3 Construction

Er den længste fase hvor de resterende dele af systemet udvikles.

### 2.2.4 Transition

Er afslutnings fasen og dette er fasen hvor et system eller program kan overleveres til kunden eller brugeren. Evt. fejl og problemer håndteres ligeledes i denne fase, det vil sige overgang til drift.

UP procesmodellen er baseret på Use cases og risici. Use cases definerer de funktionelle krav til systemet og risici identificeres ved hjælp af en risikoanalyse. Use cases prioriteres efter dem med størst risici, som skal laves først.

### 2.2.5 Hvordan er UP anderledes fra den Agile Systemudviklingsmetode ?

### 2.2.6 UP

Up er en iterativ systemudviklingsmetode hvor der arbejdes lineært med faserne inception, elaboration, construction og transition i nævnte rækkefølge. Pointen er at man i de første faser arbejder med krav og analyse og i de senere faser fokuserer på at kode og udvikle systemet indtil systemet kan leveres i transition. Yderligere skal der laves en masse dokumentation i UP inden man kan gå i gang med at udvikle hvilket passer godt til systemer hvor kravene til systemet er meget kritiske og hvor det er vigtig at kravene er defineret på forhånd.

### 2.2.7 Agilt

I modsætning til UP er agil systemudvikling ikke lineært på samme måde. I hver iteration arbejdes der med alle elementer på en gang. Det vil sige krav, design, analyse, udvikling og test. Den dokumentation man skal bruge laves løbende og der laves ikke mere end højst nødvendigt for at man kan komme i mål med færdiggørelse af systemet.

Den primære forskel på UP og den Agile udviklingsmetode er at man i UP skal lave en masse dokumentation før man kan gå i gang med at kode. Når man arbejder agilt laver man dokumentationen løbende og ikke mere end det er højst nødvendigt for at man kan udvikle systemet.



## 2.3 Scrum

Scrum er en agil udviklingsmetode, der dog alt afhængig af hvilken kontekst den nævnes i, betegnes som enten et framework til agil udvikling, eller en agil udviklingsmetode. Det der kendetegner Scrum er, at den som metode ikke beskæftiger sig de praktiske aspekter af softwareudvikling. Scrum fokuserer derimod mere på managementmæssige og projektledelsesmæssige aspekter af softwareudvikling. Dette gør, at Scrum i praksis kommer til at fungere som et projektstyringsredskab som selve softwareudviklingsprocessen styres med. Da Scrum ikke foreskriver noget omkring den praktiske side af softwareudvikling, ser man ofte i praksis, at Scrum anvendes i kombination med elementer fra en eller flere mere praksisorienterede udviklingsmetoder. Eksempelvis kan nævnes extremprogramming og unified process, der begge trods deres forskelle beskæftiger sig med den praktiske side af softwareudvikling. På denne måde kan Scrum i kombination med andre metoder danne en optimal praksis for udvikling, hvor der er fokus på både styring af projektet, samt taget højde for hvordan man vil håndtere de mere praksisnære dele af udviklingen.

### 2.3.1 Scrum artefakter

Scrum rummer tre artefakter som er kerneelementer i metoden. Der opereres med en product backlog som er den vigtigste af artefakterne. Product backlog er en prioriteret liste af krav til det system der skal udvikles. Elementerne i product backlog kaldes items, og de er i product backloggen prioriteret efter deres forretningsværdi også kaldes business value. Rent praktisk sker det, at efterhånden som der bliver fjernet items fra backloggen i takt med, at items bliver omsat til kode står man efter denne ændring med en ny version af product backloggen. På denne måde vil man komme til, at stå med en ny version af product backloggen hver gang der er blevet færdigudviklet nogle items. I kontrast til at der typisk fjernes items fra backloggen efterhånden som der udvikles mere og mere på systemet, kan det også ske at der føjes nye items til backloggen. Der kan være flere årsager til dette. Det kan være at kunden får nye forretningsbehov, der kan ske lovændringer der gør, at der skal implementeres yderligere funktionalitet på et område eller der kan for eksempelvis opstå nye konkurrencemarkeder som kunden vil satse på. Ofte er det i praksis sådan, at behovet for et stykke softwares kunnen, ændrer sig i løbet af, at det er i gang med at blive udviklet. Dette sker på baggrund af, at det scenarie som softwaren skal fungere i, kan ændre sig med korte mellemrum. Netop derfor er en af hovedtankerne bag Scrum, at man havde behov for en udviklingsmetode, der kunne håndtere ændringer i krav til produktet undervejs i udviklingsprocessen.

Det andet artefakt i Scrum er sprint backlog. Inden man går i gang med en udviklingsfase flytter man items fra product backloggen over i sprint backloggen. Dette foregår sammen med kunden eller en repræsentant for denne. Alt efter hvilken prioritering de forskellige items i product backloggen har, flyttes de mest relevante af dem over i sprint backloggen.

Det der praktisk sker når items føres fra product backloggen til sprint backloggen er, at de omsættes til user stories. Sprint backloggens formål er, at være en liste af user stories hvor hver story er nedbrudt i en række konkrete udviklingsopgaver kaldet tasks.

Det tredje artefakt er burndown chart. Burndown chart er en graf der indikerer hvor godt det lykkedes udviklingsteamet at løse de opgaver de har vurderet de kunne, indenfor den givne ramme af tid. På burndown chartet anføres der først en teoretisk linje der indikerer hvad man teoretisk set som team bør kunne brænde hver dag. Derudover påfører teamet hver dag hvor meget arbejde de har nået at lave, altså hvor meget de reelt brænder.

Burndown chart er en vigtig artefakt da den kan bruges til, at fortælle eller afsløre om teamet generelt ikke kan levere det estimerede arbejde, eller den kan lede opmærksomheden hen på om det eksempelvis er en særlig type funktionalitet, der ofte er svær at implementere til tiden.

### 2.3.2 Scrum roller – Product Owner

Scrum rummer også tre roller der er centrale i metoden. Som repræsentant for kundens behov har man en product owner. Dette kan være en person kunden selv har stillet til rådighed, eller det kan være en person internt i udviklingsvirksomheden. Product owner er den vigtigste af de roller der er i Scrum, da det er dennes opgave, at sørge for, at det der bliver udviklet faktisk er det kunden har behov for, også hvis der skulle opstå ændringer i kundens behov. Product owner er den person der kender produktet der skal udvikles og, de behov kunden har, og gerne forretningen som produktet skal varetage. Ifølge Jeff Sutherland, grundlæggeren af Scrum, har product owner nogle direkte ansvar som er følgende:

- Han har som ene person ansvar for product backloggen.
- Items i product backloggen skal være tydeligt og præcist formuleret.
- At hele tiden, og løbende omprioriterer items i product backloggen, så deres hierarki korresponderer med et virkelighedens snapshot af kundens behov.
- Hvis der er interessanter der kunne ønske ændringer i product backloggen, skal de gå til product owner, da han er den eneste person, der har ret til at ændre på prioriteringen af items.
- Han er en nøgleperson der er ansvarlig for, at definere krav og features til systemet.
- Han har til opgave, at prioritere features og dermed de items der findes i product backloggen, så de reflekterer hvad der har mest business value/forretningsværdi.
- Når nogle features er udviklet, er det hans opgave, at acceptere eller afvise de features der er blevet udviklet af udviklingsteamet. Dette sker i forbindelse med en særlig Scrum aktivitet der betegnes sprint review.

Product owner er på mange måder central, og man kan sige, at hvis ikke man har en product owner der har et solidt kendskab til forretningsbehovene, som det færdige produkt skal understøtte, kan det være svært for udviklingsteamet, at ramme rigtigt med de features de

udvikler. Product owner har som udgangspunkt, som den eneste person, ret til at varetage ovenstående, men må gerne uddelegere opgaverne til udviklingsteamet. Dog er han stadig den endeligt ansvarlige for, at opgaver udføres tilfredsstillende.

### 2.3.3 Scrum roller - Scrum Master

Den anden centrale rolle i Scrum er scrum master. Scrum master har overordnet set en serviceerende rolle overfor selve udviklingsteamet, mere end der er tale om en dikterende rolle. De primære ansvar i rollen består i, at hjælpe teamet fri af blokeringer, der hindrer dem i at arbejde effektivt, og levere de estimerede features indenfor den afsatte tidsramme. I Scrum opererer man med begrebet 'impediments', som er et udtryk der dækker over alt tænkeligt, der hindrer en udvikler i, at få leveret et stykke arbejde indenfor estimeret tidsramme. Det er derfor scrum masters rolle, at tage sig af impediments og få skabt bedre betingelser for udviklingspraksis hvis der er behov for det. Han fungerer på denne måde som en slags firewall der beskytter teamet.

Det er scrum masters ansvar, at facilitere scrum processen, via en slags projektlederrolle. Dette betyder, at han skal sørge for at scrum processen bliver overholdt. Dette indebærer, at de forskellige scrum events bliver afholdt på korrekt vis, at scrums indbyggede regler, værdier og praktikker overholdes. Scrum master kan både være en del af udviklingsteamet, men han kan i praksis også være en dedikeret person, der som scrum master kun beskæftiger sig med sikring af scrum processen. Scrum master leder det daglige Scrum møde der kaldes daily scrum, hvor alle i teamet bliver opdateret på hvordan det går andre i teamet. Generelt fungerer scrum master som et bindeled mellem teamet og product owner.

### 2.3.4 Scrum roller – Development Team

Den sidste af de tre roller er selve udviklingsteamet hvor alle udviklere i teamet under et betegnes som udviklingsteamet. Det er udviklingsteamets ansvar under hele processen at blive mere erfarende i hvordan de kan arbejde med Scrum, ligesom det er helt centralt at de får mere erfaring i at estimere tasks og vide med sikkerhed hvad de kan udvikle indenfor en given tidsramme.

### 2.3.5 Scrum Events

Scrum består udover roller og artefakter af en række foruddefinerede events, som hver især har sit formål og sin specifikke plads i scrum processen. Scrum anvender i forbindelse med events et princip om 'timeboxing', hvilket betyder at der er afsat et maksimalt tidsrum som en given event må tage. Hver Scrum Event har sit eget navn. Meningen med at have disse events er, at minimere behovet for ad-hoc møder, der antageligt vil have en forstyrrende effekt på udviklingsprocessen, samt sikre kontinuitet og fremdrift i arbejdet. En event i Scrum kan godt slutte før dens 'timebox' er brugt, hvis dens formål er opfyldt.

Det helt centrale event i Scrum er Sprint. Sprint er en fastsat tidsramme på typisk 1-4 uger, som er der hvor selve softwareudviklingen foregår. Indenfor sprintet har teamet comittet

sig til, at udvikle en bestemt mængde features, som de gerne skulle kunne nå indenfor denne periode. Sprint består af en række underaktiviteter der betegnes som, sprint planning, daily scrums, sprint review og sprint retrospective.

### 2.3.6 Scrum Events – Sprint Planning

Sprint planning er den første aktivitet der er i et sprint. Alle i teamet deltager i denne aktivitet. Formålet med denne aktivitet er, at planlægge alt det arbejde der skal foregå i det kommende Sprint. Tidsrammen for sprint planning er maksimalt 8 timer for sprint på 4 uger, men for kortere sprint er tidsrammen kortere. Formålet med aktiviteten er af finde ud af:

- Hvilke features kan vi nå at lave i løbet af dette sprint.
- Hvad for noget arbejde skal der til, for at disse features er udviklet

På dette møde tager man udgangspunkt i product backloggen. Det første der sker på sprint planning meeting er, at teamet sammen med product owner gennemgår product backloggen og drøfter de items der har størst prioritet. Med udgangspunkt i, at udviklingsteamet som skal udvikle, er dem der bedst ved hvad de kan nå indenfor en given tidsramme, er det dem der i samarbejde med product owner tager items fra den prioriterede product backlog, som de comitter sig til, at kunne færdigudvikle indenfor sprintets tidsramme. En af nøgleprincipperne i Scrum er, at teamet selv vurderer hvad de kan nå, frem for at få påtvunget arbejdsmængden udefra.

Det næste der sker er, at det gennemgås hvor meget tid hver udvikler har til det konkrete udviklingsarbejde, fraregnet eksterne møder, pauser og andet 'fravær'. På denne måde udregnes den samlede timekapacitet for udviklerholdet der er til rådighed til selve softwareudvikling. Når denne er specificeret begynder de, at nedbryde hvert item i product backloggen til tasks, som er konkrete udviklingsopgaver. Disse tasks påføres et estimeret timetal, som man skønner det vil tage, at udvikle den feature den pågældende tasks gælder for. Udviklerholdet bliver ved med, at omsætte items fra product backloggen til tasks indtil de ikke har flere timer til rådighed.

De tasks som udviklerholdet nu står med, danner tilsammen det der kaldes sprint backlog – en samlet liste over features, der skal udvikles indenfor Sprintet. Når alle valgte items er omsat til tasks i sprint backlog, kan selve udviklingen begynde. Udviklere kan herefter frivilligt påtage sig tasks, og begynde selve softwareudviklingen. Udviklerne bør dog være opmærksomme på indbyrdes afhængigheder mellem tasks, og eventuelle hensigtsmæssigt rækkefølge at implementere dem i. Når først sprintet er startet, kan product owner ikke presse nye tasks ind i sprintet. Dermed kan eventuelle nye behov, først komme med i næste sprint, ved at de blevet indført og prioriteret i product backloggen. Dog kan product owner her i ekstreme tilfælde aflyse et sprint. Alternativt kan man sige, at product owner kun er et sprints varighed eller mindre, fra at kunne kræve ændringer implementeret. For at holde

styr på product backlog, sprint backlog og arbejdets fremdrift, anvendes der ofte visuelle hjælperedskaber, der gør det muligt for alle i teamet at få overblik over hver enkelte tasks status, om den er påbegyndt, i gang, afsluttet, eller skal godkendes. Til dette formål bruges der oftest et scrum board, eller task board.

### 2.3.7 Scrum Events – Daily Scrum

I løbet af et sprint afholdes der hver morgen et møde styret af scrum master. Mødet betegnes både som daily scrum eller daily stand up meeting. Scrum master sørger for, at det kun er teamet der deltager i daily scrum. Mødet afholdes samme tid hver dag, og har samme varighed hver gang.

Hensigten med mødet er ikke diskussion, men derimod afrapportering af status for arbejdets fremgang. På mødet er der tre obligatoriske punkter alle teammedlemmer svarer på:

- Hvad lavede jeg i går, som bragte teamet tættere på vores mål for Sprintet?
- Hvad er det min intention at lave i dag, for at bringe teamet tættere på at nå vores mål?
- Er der nogle impediments jeg synes, at kunne se, enten for mig selv eller teamet?

Mødet har til formål, at kaste lys over, om teamet arbejder hen imod at nå den arbejds mængde der ligger i sprint backloggen. Daily scrum er også underlagt timebox-princippet og må tage maksimalt 15 minutter. Mødet skal også fungere, som en event der sikrer følgende:

- Forstærke kommunikation i teamet.
- Identificerer impediments så de kan fjernes.
- Mindske behov for andre møder.
- Styrke teamets viden og fremskynde dets evner til beslutningstagen.

Om mødet beskrives der også i Scrum-litteraturen, blandt andet i "The Scrum Papers" af Jeff Sutherland, at det er et 'inspicér og tilpas-møde'. Hvis der er behov for diskussion, finder det sted efter mødet.

På daily scrum mødet ajourføres teamets burndown chart gældende for sprintet. Det er vigtigt, at vide, at burndown chart ikke handler om, at vise hvor meget tid der er brugt, men derimod hvor meget arbejde der mangler, for at teamet har nået sit mål. Burndown chartet viser enten timer eller dage, eller story points hvis man kombinerer med extreme programming. Efter hvert daily scrum møde registreres der, hvilke tasks der er færdige, og det samlede antal timer for dem udregnes. Dette antal timer skal burndown chartet ajourføres med. Dette gøres ved, at tage værdien af resterende timer for den foregående dag, og føre grafen videre, til det nye antal resterende timer på den pågældende dag. Hvis

den teoretiske linje ligger over den kurve man ajourfører, betyder det man arbejder hurtigere, end man havde estimeret man ville. Hvis den teoretiske linje ligger under den der ajourføres, betyder det at man som team ikke formår, at følge med det arbejdstempo man har fastsat. Der er således 'gaps' på burndown chartet, og det er vigtigt at reagere på. Rent praktisk laver man ofte burndown chartet visuelt på et stykke papir eller karton på en væg, så alle altid har adgang til at se på det. Dog findes der elektroniske redskaber der kan generere et burndown chart.

Der er to resterende event i Scrum, som hedder henholdsvis sprint review og sprint retrospective.

### 2.3.8 Scrum Events – Sprint Review

Sprint review, som er den første der afholdes af de to aktiviteter er timeboxed til 4 timer for et sprint på 4 uger. På sprint review demonstrerer teamet, hvad de har udviklet siden sidste sprint. Typisk vil der blive demonstreret nye features, ændringer i arkitektur eller andre signifikante ændringer i produktet. Sprint review bør ikke tage mere end 30 minutter at forberede, da det mere er en demonstration af funktionalitet end et foredrag. Deltagerne til sprint review vil typisk være interessanter, og andre mere perifere personer som kan inviteres af product owner som også deltager. Udviklerteamet svarer på eventuelle spørgsmål fra andre deltagere. Det er vigtigt, at udviklerne som skal demonstrere de nye features, har haft en generalprøve inden demonstrationen for at sikre at alt fungerer.

### 2.3.9 Scrum Events – Sprint Retrospective

Sprint retrospective er en aktivitet, der foregår efter sprint review. Den er timeboxed til maksimalt tre timer. Det er scrum masterens ansvar, at aktiviteten finder sted. Sprint retrospective foregår internt i scrum teamet og formålet med sprint retrospective er selvinspektion, hvilket indebærer, at teamet reflekterer over det forgangne sprint og deres egen arbejdsproces:

- Identificere hvad der hovedsageligt gik godt, som er vigtigt at holde fast ved.
- Finde ud af hvad der gik mindre godt, som bør forbedres.
- Lave en plan for implementering af forbedringer, som der er nødvendige for teamets evne til at arbejde effektivt.

Teamets medlemmer reflekterer således over det forgangne sprint, eventuelt med udgangspunkt i noter de har taget undervejs igennem sprintet. Denne event er det sted i scrum processen, hvor teamet kan drøfte, hvad der fungerer, og hvad der ikke fungerer, og om der eventuelt skal prøves noget nyt.

For at arbejde med de eventuelle ukendte områder der kan være af et udviklingsprojekt, opererer man i Scrum med et begreb der hedder spikes. En spike er en slags proof of concept der laves på noget funktionalitet man ikke er helt sikker på om man kan implementere. Hvis der på den måde er noget meget nyt i et projekt, bør man via en spike teste på om man reelt kan implementere det ønskede. Typisk vil en spike ligge i et sprint 0, hvor det er meningen at man afklarer om man kan implementere funktionaliteten. Dog kan man godt indarbejde spikes i løbet af sprints. Man kan også komme i situationer hvor man er nødt til det.

For at give et overblik over Scrum sammenfattes der afslutningsvis med et overblik på selve processen. Denne process er samtidig et billede på hvordan man skal gribe et scrum projekt an fra start til slut. Selveste scrum processen starter med, at en kunde har et it-mæssigt behov. En masse informationer fra alle der er involverede i produktet, samles et sted, nemlig hos productowner. Denne har ansvar for, at alle disse informationer føres i product backloggen som items. Når dette er gjort, eksisterer der den prioriterede liste af items som product backloggen er. Herefter kan scrum teamet begynde på sprint planning meeting, som indbefatter, at der bliver lavet en sprint backlog ud fra de eksisterende items i product backloggen. Derefter starter det pågældende sprint, og dets længde kan ikke ændres, efter det er startet. Under Sprintet er scrum master tovholder for selve scrum processen, og der afholdes daily stand up meetings dagligt. Når Sprintet er afsluttet, står teamet med et færdigt stykke arbejde. Efter Sprintet afholdes der sprint review, hvor de i sprintet producerede nye features fremvises ved en demonstration. Til sidst i sprintet, er der sprint retrospective, hvor scrum teamet internt kan reflektere, og drøfte, hvad der gik godt, som skal holdes fast ved, og hvad der gik mindre godt eller problematiske forhold, som eventuelt skal forbedres.

## 2.4 Extreme Programming

En anden populær udviklingsmetode er Extreme Programming (ofte forkortet XP), som er blevet udviklet igennem 1990'erne. Extreme programming er en agil udviklingsmetode beregnet til at udvikle software. Fokuspunktet for Extreme Programming og agile metoder generelt er muligheden for tilpasninger løbende med udviklingen. Dette egner sig især til projekter, hvor visse elementer kan være ukendte, eksempelvis ny teknologi. Extreme Programming tager specifikt højde for ændringer i tidsplan eller såkaldte 'spikes', hvor der bruges længere tid på en opgave end estimeret.

I Extreme Programming findes der 4 hovedprincipper, som udgør grundlaget for metoden:

1. Kommunikation, hvilket understreger vigtigheden af tæt samarbejde og vidensdeling imellem kunde og udvikler. Dette er ofte opnået igennem dokumentation.
2. Simpelt, hvilket betyder, at udviklerne skal kun løse et specifikt problem ad gangen uden at forudsige eller tage højde for fremtidige problemer eller opgaver.
3. Feedback, oftest i form af forskellige typer test, herunder Unit test, Accept test mv..
4. Mod, hvilket betyder udviklerne skal være i stand til at bryde de normer og praksisser, hvis der er behov for det.

### 2.4.1 12 arbejdspraksisser

Derudover arbejder Extreme Programming med 12 arbejdspraksisser, som beskriver den ideelle udvikling.

### 2.4.2 Planning Game

Planning game er en planlægningsøvelse, som foregår i begyndelsen af hver iteration, hvor arbejdsopgaver og aktiviteter bliver estimeret og tildelt værdi. I denne proces er kunden ikke involveret.

### 2.4.3 Små udgivelser med korte mellemrum

Produktet bliver udviklet i små udgivelser med fungerende funktionalitet. Dette er til for at give kunden bedre indblik i forløbet og mulighed for bedre at komme med kritik.

### 2.4.4 Systemmetafor

Systemmetafor dækker over konceptet om at have ens navngivning terminologi, som gør det lettere for udviklerne at gennemskue programkoden.

### 2.4.5 Simpelt design

Når programkoden udvikles skal programmøren forsøge at gøre funktionaliteten så simpel som muligt.



### 2.4.6 Test

Extreme Programming binder i Unit Test, hvor testen skrives før koden udvikles, hvilket tvinger programmøren til at gennemtænke sin metode, og sikrer at koden virker når den implementeres.

### 2.4.7 Hyppig refaktoring

Da Extreme Programming kræver simpelt design og problemløsning fra dag til dag, opstår der ofte designproblemer i fremtiden, hvor den nuværende programkode ikke støtter det kommende funktionalitet. Sker dette, kræver Extreme Programming, at koden refaktoreres og udvikles mere generisk.

### 2.4.8 Parprogrammering

Parprogrammering er en programmeringsteknik, hvor to personer sidder omkring en enkelt computer og udvikler på produktet. Programmøren, som betjener tastaturet, skriver koden i detaljer, hvor den anden har fokus på det overordnede perspektiv. Der er ikke faste par i programmering, og det anbefales at bytte ofte, så alle udviklere får indblik i al koden.

### 2.4.9 Fælles ejerskab af programkode

Dette betyder, at alle udviklere har adgang til al kode og derfor også ansvar for at alt virker. Parprogrammering støtter denne praksis ved at inkludere flere personer i samme stykke programkode.

### 2.4.10 Kontinuerlig integration

I projektets forløb skal programkoden holdes opdateret til den nyeste version, for at undgå integrationsproblemer senere i forløbet.

### 2.4.11 Overkommeligt arbejdstempo

Extreme Programming fokuserer på at udviklingsholdet skal være udhvilet, og tillader derfor ikke typisk overarbejde.

### 2.4.12 Et samlet udviklingshold

I Extreme Programming opfattes kunden ikke som køber af produktet, men som bruger af produktet. Derfor er det vigtigt at kunden er tilstede under hele udviklingen og altid tilgængeligt for spørgsmål.

### 2.4.13 Fælles kodestandard

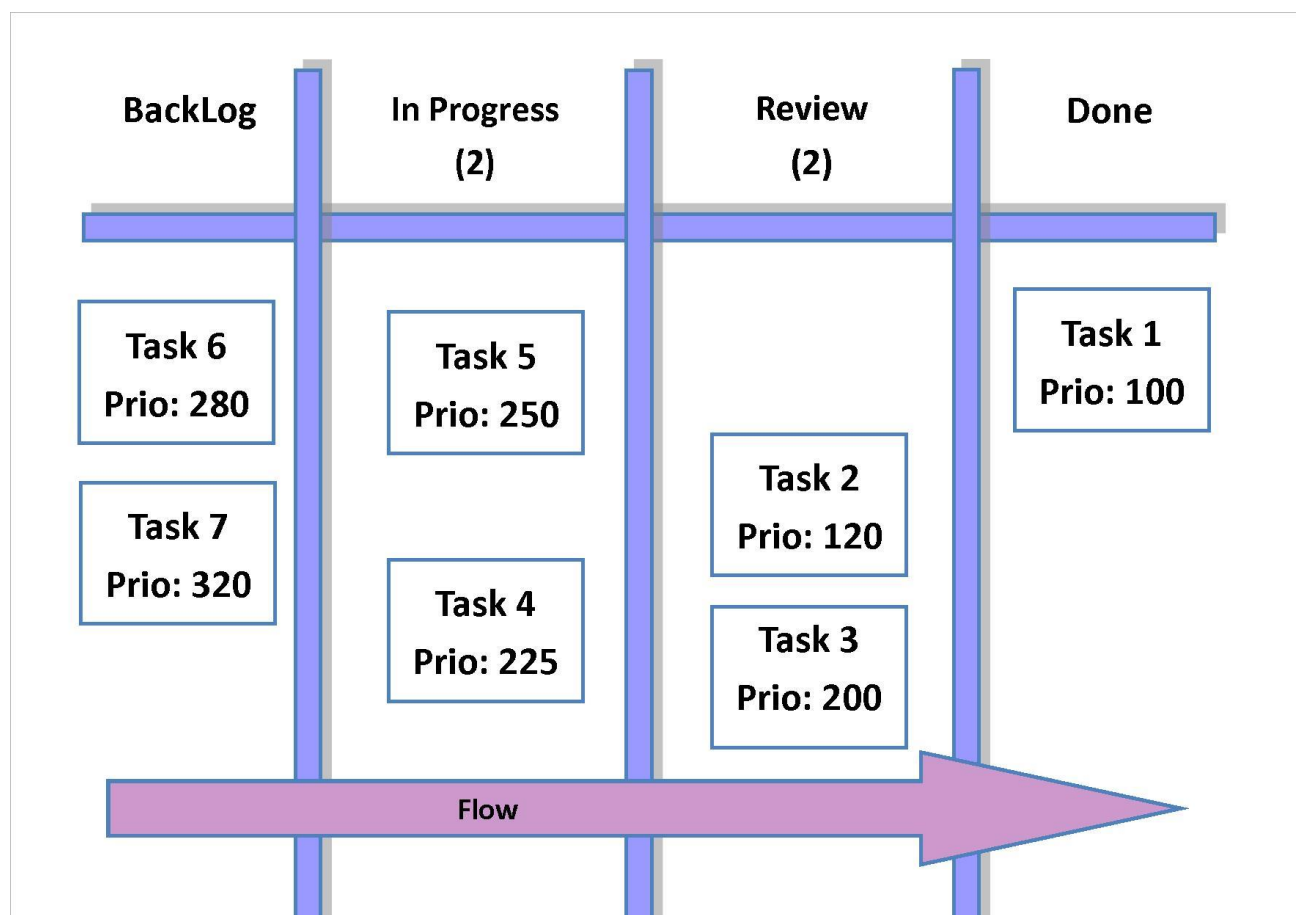
Fælles kodestandard dækker over princippet i at hele udviklingsholdet følger den samme kodestandard igennem hele projektets forløb. Kodestandarden indeholder regler og praksisser for navngivning af metoder mv..

## 2.5 Kanban

Kanban kommer fra Japan hvor Kan betyder visuel, og Ban betyder kort, og det er faktisk præcist det det er.

Det er en helt simpel metode til at visualisere og dermed effektivisere og overskueliggøre en arbejdsprocess. Således er metoden ikke indskrænket til brug for software udvikling, men benyttes også i forbindelse med f.eks tilberedning af Sushi, og fremstilling af biler.

Kanban har sit fulde fokus på flow, det handler hele tiden om, at komme videre, og hvis der opstår et problem, så er det alle mand på opgaven, så den kan blive løst, før man fortsætter. Selv om man kan bruge kanban alene, så er der tale om en letvægtsmetode, der gør sig bedst i samspil med et større framework, som f.eks. Scrum. På figur 1 kan man se et bud på hvordan et Kanban-board kunne tage sig ud.



Figur 1 - Kanban board

Umiddelbart ligner det ret meget et almindeligt Scrum-board, blot er der på kanban-boardet tilføjet tal i parantes på *In Progress* og *Review*. Disse tal er kendt som "Work in Progress"- eller WiP-limits, og signalerer klart og tydeligt hvor mange opgaver der må forefindes i den givne kolonne. I det ovenstående tilfælde ligger det f.eks fast, at for at projektet kan siges at køre på skinner, er der ingen der kan starte en ny opgave fra *backloggen*, og der er heller

ingen der kan afslutte deres *In Progress*-opgaver, før der bliver fjernet en (eller flere) opgaver fra *Review*.

På den måde kan man på en let og overskuelig måde se om ens projekt skrider fremad som planlagt. Hvis der pludselig er for mange på en kolonne. F.eks hvis en task flyttes fra *In Progress* til *Review*, så er det umiddelbart klart for enhver i teamet, at noget er skævt, og alle kan så samles og bidrage til at løse kniben.

Det er vigtigt, at WiP-limits bliver fastsat efter antal udviklere i teamet. Hvis de sættes for lavt, er der risiko for, at der er teammedlemmer som bliver låst inaktive, fordi der ikke må startes nye opgaver. Den slags forøger produktionstiden (såkaldt: "lead time"). Hvis man sætter WiP for højt, kunne man lige så godt lade være med at bruge kanban, da eventuelle flaskehalse alligevel først opdages på et senere tidspunkt.

### 2.5.1 Scrum + Kanban

At anvende kanban i et Scrum-projekt kunne afhjælpe uklarheder ifb med udviklingen. F.eks kan man indsætte et kanban-board i sit Scrum-board, og på den måde indsætte flere trin i for eksempel *Igang*-kolonnen for at repræsentere de trin *igang* dækker over, og samtidigt få nytte af WiP.

Selvom brugen af Scrum til en vis grad sikrer isolering af opgaver, i.e. det er givet hvilke User Stories der forventes løst i løbet af et sprint. Så er det nærmest umuligt at undgå, at skifte frem og tilbage mellem de mindre tasks. Det er især gældende hvis man samtidigt anvender XP-praktikkerne. Det kan give flaskehalse i løbet af et sprint, men ved at have WiP implementeret inde i selve *doing*-kolonnen, ville man kunne bevare overblikket på task-niveau

Samtidigt ville det hjælpe en eventuel projektleder, eller product owner til at forstå hvor i processen en given opgave befandt sig.

Slutteligt forhindrer en implementeret kanban halvfærdige opgaver. Pga. WiP, er man konstant tvunget til at levere 100% færdige opgaver. Scrum i sig selv kræver blot færdige opgaver på den yderste dag af et sprint.

## 2.6 Arkitektur

Scrum foreskriver, at man laver lige præcis så meget design forarbejde som er påkrævet. Hvad et udviklingshold så føler der er "påkrævet" vil givetvis være helt forskelligt fra virksomhed til virksomhed. Det er dog forbundet med en række fordele, at man overvejer en grundlæggende struktur fra starten af et projekt. Som i et traditionelt plandrevet projekt, kan man med fordel tale med kunden. Det produkt som denne ønsker, bør lægge til grund for den grundlæggende arkitektur.

Der er flere praktiske grunde til, at det kan være en fordel at have en basal arkitektur fra starten. Det kan hjælpe udviklerne til, at estimere projekts størrelse, både i tid og penge. Typisk er der nogle interessenter, være det så banken, product owner, eller virksomhedens ledelse, der ikke har kapacitet til, at forstå de udfordringer et projekt består af. Hvis man har et tentativt design, så har man også bedre muligheden for at kommunikere disse udfordringer igennem fra start. Derudover kan det også vise sig, at være et nyttigt holdepunkt senere i projektet hvis der opstår tvivl om retningen.

## 2.7 Metodevalg

Det er op til hver enkelt virksomhed, eller udviklerteam, om de mener en plandreven, eller agil fremgangsmetode er passende. Passende både for teamet der skal udføre det, men også for den type projekt som skal udvikles.

Tidligere i rapporten er beskrevet de karakteristika som man kan forvente af hver type, men hvilke parametre kigger man på når valget skal træffes?

Boehm specificere bla. de følgende kriterier:

- Hvor kritisk en fejl er - Hvor slem er konsekvensen?
- Dynamik i omgivelserne - Hvor tit kommer der nye krav?
- Udviklere - Hvor erfarne, og hvor mange, er de?
- Kultur – Er virksomheden, og udviklerne, gearet til den valgte metode?

Jo mere kritisk en fejl ville være for brugeren, jo vigtigere er det at der er lagt fokus i at imødegå fejlen. I et plandrevent projekt er der, i hvert fald i teorien, bedre muligheder for at undgå kritiske fejl, da alt er veldokumenteret. I et rent agilt forløb er udviklerne i højere grad afhængige af et solid fælles kodekendskab, og god kommunikation.

”Dynamik i omgivelserne” refererer til hyppigheden hvormed der introduceres nye krav og/eller ændringer til projektet. Agile projektgrupper er bedre gearet til at imødegå nye tiltag, da hvert sprint er at betragte som en unik enhed af det samlede projekt. Her er en plandreven gruppe mere presset, da man i værste fald skal tilbage til analyse fasen for at kunne integrere noget nyt.

Størrelsen på et udviklingsteam er en væsentlig faktor når det kommer til at vælge projektform. Den høje grad af dokumentation gør, at et stort team har mere ud af at benytte traditionelle plandrevne metoder. Omvendt ville et stort team gøre kommunikationen, og den uformelle dokumentation der hører til agil udvikling, til et mareridt.

Samtidigt ligger der en væsentlig faktor i, om hvorvidt udviklerne er erfarne nok til at kunne håndtere den agile arbejdsmetode. Det kræver en høj grad af overskud og selvdisciplin at følge f.eks. XP praksisserne. Det er selvkært, at jo mere erfaren og dygtig et udviklerhold er, jo bedre muligheder har et projekt for at blive gennemført på en god måde. Men den høje grad af dokumentation i et plandrevent forløb giver en fordel til den uerfarne udvikler, der stadig er igang med at lære faget.

Sidst, men ikke mindst, er der virksomhedens kultur, at tage højde for. Der er ikke så lidt forskel på den måde man arbejder på i de forskellige metoder. Derfor kan det komme som et chok for de ansatte som måske altid har været vant til at arbejde plandrevent, hvis de pludselig kastes ud i agile forløb. Derfor er det væsentligt, at tage temperaturen på alle dele af virksomheden for at se om det overhovedet er realistisk at starte et agilt forløb.

Baseret på disse faktorer, og evt. flere som man selv definerer, kan man lave et såkaldt "Projektkort", som kan anvendes til at træffe beslutningen.

## 2.8 Risikostyring

En "Risiko" defineres som: "En mulig negativ planafvigelse". Altså, en hændelse der kan bringe vores tidsplan, rygte, og i sidste ende, vores job i farezonen.

For bedst muligt, at kunne ruste teamet imod konsekvenserne af en opfyldt risiko, er det vigtigt, at kunne komme disse risici i forkøbet.

Risikostyring/analyse dækker over den valgte metode til at identificere, og håndtere, de problemer der eventuelt måtte opstå i forbindelse med afviklingen af et projekt. Selv i mindre projekter er der en lang række ting der kan gå skævt, og derfor er det også vigtigt, at kunne prioritere de hændelser man forudser.

### 2.8.1 Risikostyring i Scrum

I et klassisk plandrevet scenarie udfører man en grundig risikoanalyse i starten af et projekt, eventuelt med tilhørende prototyper etc. Det kan man naturligvis også gøre i et Scrum-projekt, og i så fald foregår det i det såkaldte "Sprint 0".

En alternativ måde at tilgå risikostyringen på er ved, at tage det på et sprint til sprint basis.

Hvor man for hver afsat timebox i projektet vurderer hvilke risici der lurder i horisonten, og så håndtere dem som en naturlig del af det kommende sprint.

En tredje måde at håndtere risiko i et projekt, er at vælge et trin imellem disse to yderligheder. Det vigtige er, at man tilpasser sig til situationen.

### 2.8.2 Kvalitetssikring

I modsætning til plandrevne projekter, er der i den agile ikke nogen fastlagte processer ifh. til kvalitetssikring. Det vil sige, at ligesom med risikostyring, arkitektur, og det meste andet, så er det op til det enkelte udviklingsteam at fastlægge sine egne procedurer.

Scrum kommer selv med et par indbyggede muligheder for, at kvalitetssikre sit produkt.

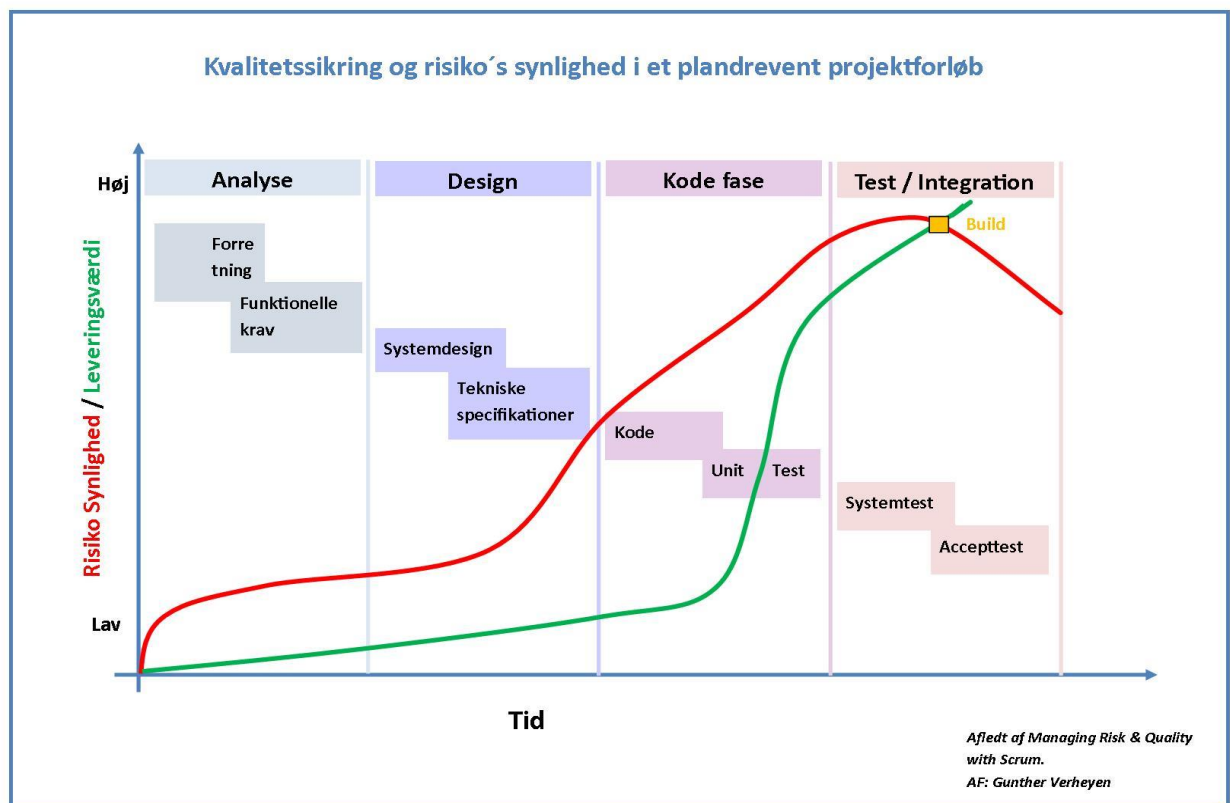
Hver user story kommer med en, af product owner defineret, accepttest. Acceptestens krav, samt informationerne på user story'en danner udgangspunkt for, at skrive de UNIT tests der

udgør Test First-praktikkens indhold. Derudover er der review indbygget i hvert sprint, så man ofte får rundet produktets kvalitet med product owner, som er den yderste autoritet på området.

I forhold til at kvalitetssikre koden kan man med fordel skæve til de 12 XP praktikker (og 4 værdier), da de alle, i større eller mindre grad støtter op om at sikre en god kodekvalitet. I den forbindelse kan det godt betale sig at være opmærksom på, at såfremt man vælger en eller flere af disse praktikker fra, så opstår der huller i det "sikkerhedsnet" som alle praktikkerne tilsammen udgør. Hvis et team f.eks. fravælger parprogrammering, så bør man sikre sig, at man istedet indfører et kodereview, da man ellers er åben overfor fejl.

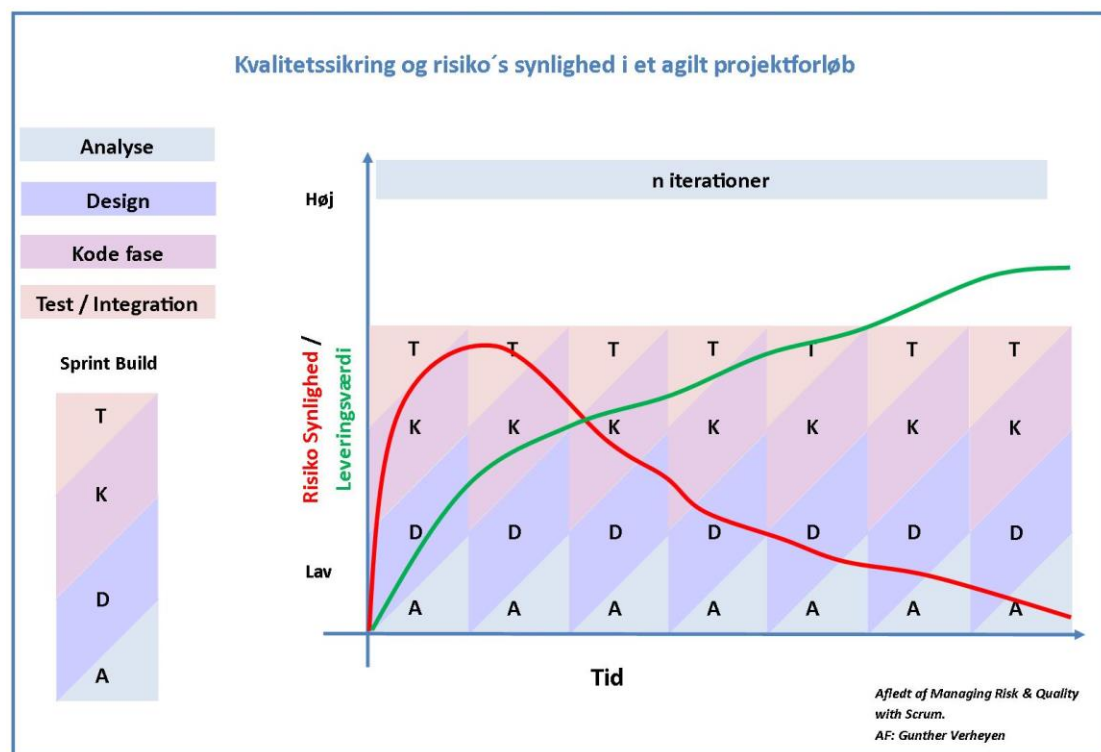
Samlet set bør man dog være bedre rustet til at opnå et godt kvalitetsniveau, hvis man anvender en Scrum + XP drevet metode fremfor f.eks. UP. Det er naturligvis noget af en påstand, og det forudsætter da også, at man overholder de praktikker der er fastsat (eller har et solidt sikkerhedsnet hvis man vælger nogen af dem fra). Men hvis man gør det, så sikrer man ved hver eneste user story, at såvel de formelle, som de uformelle kvalitetskrav er opfyldt.

I figur 2 og 3 kan man se forskellen mellem plandrevet og agil udvikling i forhold til kvalitetssikring og risikostyring. Som man kan se opnår man ved brug af agil udvikling, både den fordel at kunne se risici langt tidligere i projektet, og produktet får langt hurtigere en høj leveringskvalitet. Bruger man en plandreven metode, risikerer man først at kunne se en risiko når man nærmer sig projektets afslutning. Samtidigt er det også først hen imod slutningen af et projekt, at produktet bliver samlet så det kan leveres.



Figur 2 - Plandrevet kvalitet og risiko





Figur 3 - Agil kvalitet og risiko

## 2.9 Krav Specificering og Vedligeholdelsesstyring

Kravsspecificering for et produkt skal ofte foregå på et meget detaljeret plan, så det er tydeligt hvad det enkelte krav indeholder. Krav kan både danne ramme for kontrakter og er på samme tid rettesnoren for designfasen i et projekt. Krav gøres op i funktionelle og ikke-funktionelle krav. Ofte er det de ikke-funktionelle krav, der er mest kritisk at få tydeliggjorte, da de i værste fald kan gøre systemet ubrugeligt, hvis de fejler. Endeligt kan de ikke-funktionelle krav deles op i følgende tre grupper: Produkt og kvalitetskrav, organisatoriske krav og eksterne krav. Organisatoriske krav vil være bundet til, hvordan omstændighederne er i organisationen. Eksterne krav kan være eksempelvis lovgivning eller integration med andre it-produkter.

I praksis kan kravspecifikationen være et konkret dokument, men når man arbejder agilt, vil de funktionelle krav typisk være repræsenteret, via product backloggen, hvor de øverste krav vil være dem der er mest detaljeret beskrevet. Da agile metoder er modstander af formel dokumentation, kan man sige at product backloggen fungerer, som en slags inkrementel dokumentation, hvilket passer fint med, at man hen gennem den agile process, løbende kommer til at stå med nye versioner af product backloggen. At specificere krav er både komplekst og tidskrævende. Der kan spille mange forhold ind, der komplicerer specificeringen af krav. Disse forhold kan blandt andet være lovgivning, ændringer i kundebehov, underleverandørers ændringer i underprodukter og endelig kan det være, at kunderne ikke er sikre på hvad de præcist vil have, eller at de ikke præcist formår at give udtryk for det, i relevante termer der er it-faglige eller it-relaterede.

Agile metoder er baseret på løbende kravændringer, og derfor skelnes der ikke mellem nyudvikling og videreudvikling, som også kaldes vedligehold. Praktisk idriftsættelse af ny funktionalitet er det kunden oplever som en ny version af produktet. Typisk sker videreudvikling ved, at der enten tilføjes komponenter, eller laves ændringer i eksisterende komponenter. Grunden til vedligeholdelse er vigtigt er, at det oftest vil være for dyrt og risikabelt, at implementere nye systemer til at erstatte de gamle, frem for at udvikle på det der allerede er integreret. I forbindelse med vedligehold er der ofte en række forhold, der har indflydelse på omkostningerne for vedligeholdelse. Det er afgørende om der er stabilitet omkring teamet forstået på den måde, at hvis vedligeholdelsen overgår til nye udviklere, vil der gå viden og effektivitet tabt. Udvikleres erfaringsniveau har også betydning, da det kan afgøre, hvor meget domæneviden de har indenfor systemets område. Endvidere kan det være, at systemet ikke fra starten er designet optimalt til at blive ændret på. Endeligt kan der være tale om, at systemet er skrevet i legacy-sprog, eller der kan mangle dokumentation, eller der kan være manglende konsistens i koden.

Når man videreudvikler på et system, vil man typisk foretrække re-engineering, da det er billigere end at udvikle et helt nyt system. Et eksempel på en moderne metode der faciliterer re-engineering, er extreme programming, som netop taler om refactoring, som en

af sine 12 praktikker. Dette kunne være et argument for at integrere extreme programming i sin udviklingspraksis.

## 2.10 Deployment og Configuration Management

### 2.10.1 Deployment

Deployment er fasen hvor man sætter sit produkt i drift og det er den sidste fase i udviklingen af et system. Det gælder både for UP, hvor det drejer sig om transition fasen, i scrum under releasing fasen og i XP under produktionsfasen. Præ-betingelsen for at gå i gang med deployment er at man har implementeret et funktionelt system som er testet.

Strategien man anvender i agile udviklingsmetoder er incremental deployment. Det vil sige at man deler funktionalitet op i releases og udgiver dem løbende startende med den første release. For at kunne styre disse releases laves en release plan. En release plan i agile systemudviklingsmetoder er storydriven, det vil sige at kunden vælger de stories der skal med i en release. Planen er også date-driven hvilket vil sige at der fastlægges et antal release datoer og releases deles op i iterationer/timeboxes. Det tilstræbes i øvrigt at lave så små releases som muligt.

### 2.10.2 Deployment fase

Accepttest i agile udviklingsmetoder er på user story niveau, det vil sige at der skal accepttestes løbende under udviklingen. En user story er ikke færdig før dens accepttest er bestået. Hvis man følger de agile idealer betyder det så at der ikke skal være et deploymentsprint til den afsluttende fase af udvikling af et system. Men dette er dog sjældent muligt. Derfor laves der også en endelig accepttest af releasen før den kan frigives.

### 2.10.3 Configuration management

For at kunne håndtere ændringer til systemet efter idriftsættelse af release 1 laves der konfigurations styring. Ved ændringer af systemet skal der etableres en ny version af systemet. Det kan for eksempel være versioner med forskellig funktionalitet eller versioner der kører på forskellige devices. Dette gør det også muligt at lave specifikke versioner til specifikke kunde krav. Derfor er det en del af kvalitetssikringen at systemændringer styres på denne måde.

## 2.11 Product Vision

Det følgende kapitel omhandler det konkrete produkt som vi har til hensigt at udvikle i løbet af projektperioden.

### 2.11.1 Navn og Applikation Statement

Applikationens navn er Level-Up.

Applikation statement : En personlig træningsassistent hvor man kan komme i form på en underholdende måde.

### 2.11.2 Behov

Vores produkt vil opfylde behovet for at have det sjovt med at træne. Dette gøres ved at lave træning om til et spil. Websiden vil give brugeren belønninger i form af xp, level up og achievements. Brugeren vil blive glad hver gang han eller hun får achievements og level up. Websiden vil gøre det muligt for brugeren at konkurrere imod andre brugere af websiden, hvilket vil tale til deres konkurrence behov.

### 2.11.3 Produkt

De vigtigste features for produktet:

- En bruger kan registrere sin træning
- Brugere af websiden kan sende en battle invite til en ven
- Brugere kan få xp og point for træning
- Brugere kan få achievements

Det unikke salgs argument for vores produkt er at den skiller sig meget ud fra andre lignende produkter ved at være baseret meget mere på spil og det at have det sjovt, samtidig med at brugeren får motivation for at træne noget mere.

### 2.11.4 Værdi

De vigtigste features for produktet:

- Applikationen vil ikke skabe et nyt marked men udfordre det marked der allerede eksisterer for træningsapps.
- At udvikle vores brand.
- Det kan skabe den værdi, at har vi først fået skabt en brugergruppe kan vi udvide forretningsmodellen til fitnesscentre og større aktører.

### 2.11.5 Konkurrence

De vigtigste features for produktet:

- Hovedkonkurrenter er Endomondo og Runkeeper, og på et internationalt plan apps som Fitocracy.
- Disse apps er lokale apps på den device hvor de ligger. Vores applikation er et website.
- Disse apps har en klassisk UI, hvor vores vil have en mere spillende UI.
- Styrker :
  - Vi antager at en hel klar styrke ved vores applikation er at den indeholder en mere underholdende og fængende brugeroplevelse end alle konkurrerende apps.
  - Når motivation er en afgørende faktor i individuel træning, tror vi på at en brugerfladen og brugeroplevelsen kan have stor indflydelse på succesraten. På denne måde antager vi at den spillende brugerflade vil kunne forære 'gratis' motivation til brugere af applikationen.
  - Samtidig er der nytænkning i at anvende et brugerflade, 'Look and Feel' fra gamerkulturen i en kontekst hvor det ikke er set før, her personlig træning.
- Svagheder :
  - Det kan være svært at overbevise person om hvorfor de skulle bruge et website til at støtte deres træning når alle andre anvender apps til det. Dette er dog kun gældende frem til, at vi kan udvikle egen app.
  - Det kan være meget svært at få plads på markedet i forhold til de eksisterende populære apps,

### 2.11.6 Målgruppen

Den primære målgruppe er folk, som mangler motivation for at træne og som synes at det er sjovt at konkurrere i spil med andre mennesker.

Er i udgangspunktet alle som har lyst til at starte, eller fortsætte deres træning. Men måske mangler motivation, eller bare synes at det kunne være en sjov måde at tracke sin træning på. Motivationen består dels i, at de måske ikke dyrker en hold eller konkurrencesport, alligevel bliver præsenteret for konkurrenceelementer. Brugen af achievements taler til samleren i os, og de forskellige variationer over temaet XP giver en konstant følelse af fremgang.

Implementeringen af et liga-system kunne sikre at medlemmer på alle træningsniveauer ville kunne finde passende modstandere at udfordre.

Selvom det principielt ikke giver mening at ekskludere nogen pga. deres alder, vil det måske alligevel være at foretrække med et lidt mere modent publikum, da det forhåbentligt kunne mere medvirkende til at forhindre snyd. Selvom konkurrenceelementet er vigtigt, er det meningen at det skal være sjovt, mere end konkurrence.

#### 2.11.7 Brugsscenarier

De følgende scenarier er enkeltstående elementer af produktet, som vi kunne tænke os, at implementere. Som sådan er de forløbere for vores user stories. Det er altså ikke nødvendigvis givet, at alle scenarierne kommer med. Det vil være op til vores product owner, at prioritere de væsentligste.

#### 2.11.8 Login

Brugeren skal være i stand til oprette en bruger og logge ind via brugernavn/password. Når brugeren opretter sig, skal han (m/k) give de følgende oplysninger:

- Navn & personlige oplysninger
- Højde, vægt, fedtprocent, kondital, alder, mål, kropstype
- Træningstyper - Interesser

Brugeren gemmer sine oplysninger, og er oprettet i systemet.



### 2.11.9 Log træning:

- Brugeren navigerer til den korrekte skærm
- Brugeren opretter en entry i logbogen, hvor han vælger eksempelvis "løb, 30 min, 4km" og gemmer denne.
  1. Brugeren får en eXperience Point (xp)-værdi tildelt på baggrund af entry'en.
  2. Det kan resultere i level up, og/eller achievement, title
  3. Systemet gemmer forbrugt kalorie etc.

### 2.11.10 Ved level up

1. Systemet kigger om der er xp nok til et level up hver gang brugeren får tildelt xp
2. Systemet kigger om ens level udløser en achievement
3. Brugeren får en visuel respons om levelup. Nye dele af systemet bliver måske gjort tilgængeligt?

### 2.11.11 Profiloversigt:

1. Brugeren kan se personlige statistikker over totalt forbrugt kalorier, ugentligt motion mv.
2. Brugeren kan se personlige achievements, level, optjent xp, Avatar, title mv.
3. Brugeren kan indstille sin konto
4. Brugeren kan indstille hvilke typer challenges han vil modtage

### 2.11.12 Leaderboards:

1. Tracker hvem der er bedst til sport.
2. Flere ligaer, og rangordninger baseret på brugerens niveau.

### 2.11.13 Challenges:

1. Det er muligt for brugeren, at udfordre andre brugere til dyst.
  1. Eksempelvis hvem der løber flest kilometer på en uge.
  2. Achievements, xp mv. til vinderen.
3. Weekly Challenges
  1. Run from Zombies! (Løb 15 min)

### 2.11.14 Statistik:

1. Systemet skal tracke brugerens fremskridt (og tilbageskridt)

## 2.12 Kanvas

Som redskab til, at gribe an hvordan vores ide kan gøres til en forretning, har vi anvendt Business Canvas, til at lave en slags kortlægning af, hvordan produktet dels kan markedsføres, og hvordan der kan tjenes penge på det. Der er også gjort tanker om partnere og potentielle kunder.

### 2.12.1 Værdiskabelse

Helt basalt er value proposition for produktet, at det skal være en anderledes 'Feel-Good' oplevelse, at registrere sin egen træning end man kender fra lignende applikationer. Det der kendetegner vores applikation er, at den skal være en trænings-registreringsapp, der rummer et gaming-aspekt, hvilket blandt andet i denne tidlige version, understøttes af en spillignende brugerflade. Dette er et segment af applikationen, der ville skulle sættes meget på, i forbindelse med udvikling af en færdig version, da det er her, vi vil kunne få produktet til, at skille sig ud på markedet. Der skal være en underholdningsværdi ved at bruge applikationen til at registrere sin træning, hvor formålet netop er, at brugeren via interaktion med applikationen ikke kommer i kontakt med en typisk, administrativt lignende brugerflade. Dette skal give motivation for at træne og bruge applikationen. På denne måde skal det være sjovt at komme i form.

### 2.12.2 Kundesegment

Kundesegment vil være gameren der gerne vil træne, eller personer der generelt er tiltrukket af et gamificeret forhold til sin træning. Gamere er der rigtig mange af, og derfor en målgruppe der kunne være oplagt at fokusere på.

### 2.12.3 Nøglepartnere

Nøglepartnere vil være webhostingselskaber der hoster vores applikation og brugerdatabase. Andre partnere kunne være LAN-organisationer, eller arrangører af gamingevents der har forbindelse til rigtig mange gamere, i kraft af de events de arrangerer. Man kunne satse på, at lave en eventuelt abonnementsaftale med fitness-centre, der kunne udbyde denne applikation til deres medlemmer. På denne måde vil vi være sikret, en løbende indkomst hvor fitness centret er vores kunde, som ved at de abonnerer på vores applikation, kan tilbyde brug af applikationen til deres medlemmer. Dette vil være en oplagt markedsvej at satse på, da det alternativt kan være meget svært at blive set i app stores.

#### 2.12.4 Primære ressourcer

Primære ressourcer vil være servere, udviklere til udvikling af applikationen og nye features med tiden. Det vil derudover være vigtigt med designere, da det designmæssige aspekt af denne applikation vil være af afgørende karakter og betydning, da hele value proposition meget kommer af den oplevelse, brugeren får via interaktion med brugerfladen.

Det er vores hensigt, at denne applikation skal bygge på en service der kører på en server. Denne service vil derefter kunne udstille applikationens funktionalitet, til de klienttyper vi gerne vil udvide produktet, til at kunne bruges på. Som udgangspunkt starter applikationen med at være et website, men ved at have implementeret selve applikationslogikken som en service, er det forholdsvis nemt at lave apps til eksempelvis Android og iPhone der via netværk konsumerer denne service. På denne måde kan vi udvide konceptet med tynde klienter. Der skal også lægges en del kræfter i markedsføring, da det er vigtigt at få denne applikation gjort synlig for potentielle brugere.

#### 2.12.5 Kunderelationer

Selve kunderelationen varetages gennem, at der skal være en community integreret i applikationen. Derudover er det et vigtigt aspekt, at brugeren oplever troværdighed i produktet, ved at se resultater ved at bruge det. En anden kunderelation end den hvor vi har direkte forbindelse til slutbrugeren, er som omtalt, at få etableret abonnementsaftaler med større organisationer som eksempelvis fitnesscentre.

#### 2.12.6 Kanaler

De kanaler vi tænker at anvende for at blive set, er for eksempel gamingsites på nettet, portaler og forums hvor gamere findes. Derudover kunne det være oplagt, at blive set på gamingevents, og eventuelt have en subevent der omhandlede brug af applikationen. Endelig er der den mere kommercielle vej til brugeren gennem reklamer i magasiner.

#### 2.12.7 Omkostninger

Udgifterne i forbindelse med at gøre produktet til en forretning, vil primært bestå i udgifter til at udvikle produktet. Derudover vil der også skulle anvendes en del midler på markedsføring. Det optimale her ville dog være, at kunne slå igennem ved, at blive markedsført viralt, og eventuelt blive set på de sociale medier.

#### 2.12.8 Indtjeningsmuligheder

Indtjeningsmulighederne vil være struktureret således, at der skal være en gratis version af applikationen, som det ikke koster noget at bruge. Den vil dog indeholde reklamer, som brugeren i en udstrækning vil blive mødt af. Derudover skal der være en abonnementsversion som kunden betaler en fast pris for, for en given tidsperiode. Endeligt vil det være oplagt, at udvikle produktet til flere platforme for, at give kommende brugere af applikationen mulighed for, at tilgå applikationen fra den device de ønsker.

### 3 Sprint 0

I et projekt, som benytter agile metoder, er der oftest gjort brug af en såkaldt 'iteration 0', hvor al forberedelsen inden projektets reelle start tages hånd om. Dette betyder denne iteration oftest er en kritisk fase for resten af projektets forløb, da denne danner grundlag for både planlægning, design og produktet.

I dette projektforsløb var gruppen fastbesluttet på at gøre stort brug af en iteration 0, da et tidligere projekt led under manglen på denne. Derfor blev den første uge, som var til rådighed, brugt til dette formål. Iterationen bestod af mange elementer, hvoraf størstedelen var design og udarbejdelse af idéer til produktet. Sidst i ugen bestod arbejdet i at arbejde med spikes.

Om mandagen d. 25/11, som var opstarten på projektforsløbet, stod programmet på brainstorm og idégenerering, hvor vi på klassen arbejdede igennem flere brainstorming metoder. Disse metoder resulterede i adskillige idéer, og efter filtrering skulle de bedste idéer fremlægges for klassen. Efter fremlæggelserne skulle alle elever stemme på deres favoritidé fra hver enkelt gruppe, hvilket resulterede i den endelige idé, som blev grundlaget for projektet. Idéen blev udspecificeret, der blev lavet de første mockups på papir, som skulle hjælpe med at illustrere funktionaliteten.

Dagen efter gik med emnet prototyping, hvor hovedfokus lå på persona og målgruppe. Der var delte meninger omkring emnet, hvor nogle gruppemedlemmer mente, at applikationen i princippet var for alle typer mennesker, hvorimod andre mente, at nogle personas var bedre egnet som målgruppe end andre. Der blev derfor i samarbejde med en forelæser udarbejdet 2 personas, som skulle vise de to ekstremer i målgruppen. Dette var et vigtigt element i udviklingen, da det støttede designprocessen i at have en forståelse for, hvordan produktet eventuelt skulle markedsføres.

Om onsdagen var der besøg fra en ekspert i området, hvor alle grupper fremlagde deres idé og fik efterfølgende feedback. På den baggrund fik vi klarlagt hvordan produktet differentierer sig ift. konkurrerende produkter.

Sidste del af iteration 0 var der fokus på de spikes, som gruppen havde forudset kunne volde problemer. Dagene bestod derfor udelukkende af vidensindsamling og eksperimentering af teknologi. Specifikt blev der lagt vægt på at få forståelse for REST-service, som skulle være en kritisk komponent i produktet, og da denne teknologi var ny for alle gruppemedlemmer tog denne del mange ressourcer. Samtidig blev der brugt tid på at undersøge og lære Entity-framework, som har været et værdifuldt værktøj i processen.

#### 3.1 Retrospective

Iteration 0 gavnet gruppen rigtigt meget, da det blev afsat tid til at vurdere og diskutere de overvejelser, der var blevet gjort sig under designfasen. Samtidig gav det mulighed for at

undersøge og vurdere de nye teknologier, som eventuelt skulle bruges i projektet. På denne måde undgås risikoen for spildt arbejde, hvis en teknologi måtte droppes.

Ugen afsluttede med at udarbejde produktbackloggen, hvor alle features var prioriteret. I udarbejdelsen af denne har der været fokus på at prioritere de mest essentielle funktioner højest.

Det første udkast af listen ses her:

ID	Navn	Beskrivelse	Prioritering
1	Tilføj log	Bruger: Oprette entry i træningslogbogen	100
2	Progress tab	Bruger: Se fremgang i form af LVL, XP, Ach...	200
3	Beregning af XP	System: Skal balancere opbygning af XP	300
4	Tildel Achievements	System: Skal tildele achievements	400
5	Opret bruger	Bruger: Oprette sig i systemet	500
6	Admin egen profil	Bruger: Skal kunne administrere privatindstillinger	600
7	Daily/Weekly quest	Bruger: Løse fx daglige opgaver	700
8	Quests	System: Liste af løselige opgaver	800
9	Tilgå tidligere log	Bruger: Holde styr på eksisterende entries	900
10	Leaderboards	System: Rangere brugere efter WoP's (WorkOutPoints)	1000
11	Udfordr andre	Bruger: Udfordr en anden bruger til træning	1100
12	Online forum	System: Forum til diskussion, vejledning mv.	1200

Igennem processen er der tilføjet flere features eller en feature er splittet op i flere:

ID	Navn	Beskrivelse	Prioritering
13	REST-Spike	System: REST-service, forbindelsen mellem client og backend	250
14	Login med Bruger	System: Åbne en bruger via brugernavn og password	550
15	Nyligt oprettede logs	System: Vise brugeren den nye log som er indtastet	950
16	Login sikkerhed	System: Hashe brugerens username og password	450
17	Ændre kodeord	Bruger: Er i stand til at ændre sit kodeord	650

## 3.2 Metodevalg

Ved slutningen af sprint 0 var teorien bag metodevalg sat i brug. Resultatet ses herunder:

### 3.2.1 Criticality

Selvom det ville være brandærgerligt om hele projektet kuldsejlede, så er der dog næppe grund til at frygte at nogen mister livet herved. Det mest kritiske resultat vi kan opnå, ville være kompromitteringen af de personlige oplysninger som en bruger indtaster. Det kan i bedste fald påvirke vores komfort, og i værste fald resultere i formindsket indtjening.

### 3.2.2 Culture

En intern vurdering har klarlagt at  $\frac{3}{4}$  af teamet identificerer sig selv som tilhørende order, og i udgangspunktet bedst trives med den plandrevne projektmetode. mens sidste medlem fungerer bedst på kaos, og altså umiddelbart foretrækker en agil løsning. Derfor ville et

team som vores ikke nødvendigvis vælge en agil projektmetode, uden udefrakommende påvirkning.

### 3.2.3 Dynamism

Vi har lavet et udemærket forarbejde på projektet, og er enige om det generelle koncept. Da vi samtidigt har product owner som medlem af gruppen er det rimeligt at antage, at der ikke kommer mange uforudsete krav. Her gør det sig også gældende at projektet har en ganske kort tidshorisont. En af de vigtigste grunde til at arbejde agilt, er at kunne håndtere skiftende kundespecifikationer på en smidig måde. Hvis der ikke kommer så mange ændringer, så kunne man med fordel overveje at bruge en plandreven metode istedet.

### 3.2.4 Technology

Vores eget bidrag til modellen repræsenterer den technology vi har til hensigt at anvende. Vi forventer som nævnt i Dynamism, ikke ret mange ændringer i det overordnede produkt. Derimod har vi en klar forventning om, at vi kan komme ud for at skulle spike på ny teknologi igen og igen. Det skyldes at flere af de værktøjer vi skal anvende er helt ukendte for os. I et plandrevent projekt hvor man f.eks. anvendte UP, ville der være afsat tid til at lave prototyper i starten af projektet. Men derefter ville det være en proces op ad bakken at finde tid til uforudsete spikes. Da vi i teamet ikke har mange års erfaring i udvikling af nye systemer, antager vi at uforudsete spikes, er en ting der meget let kan opstå. Derfor placerer vi os tæt på centrum af akse, da en agil udviklingsmetode både giver muligheden for, at placere spikes midt i projektet, og for at kunne forventningsafstemme med product owner.

### 3.2.5 Personnel

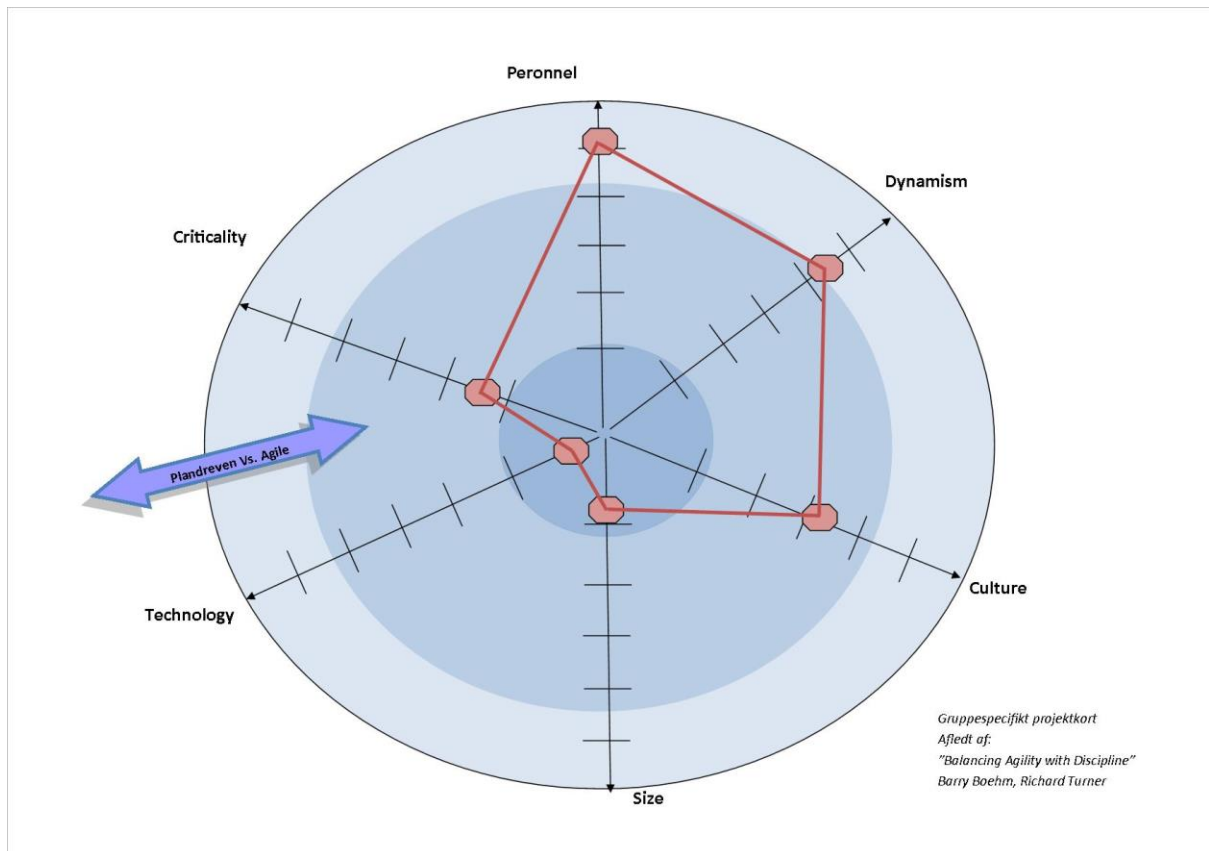
Der er ikke nogen afgørende forskel på teammedlemmernes kompetencer. vi anser os selv som værende ganske kompetente for udviklere på vores erfaringsniveau. Medlemmerne i gruppen er både i stand til, og villige til, at tilpasse en metode til de situationer der opstår. Man kunne med en vis ret argumentere, at vi mangler den erfaring der skal til for at kunne placere os på niveau 2 eller 3. Og derfor har vi valgt Level A1. Det vil sige et niveau hvor vi kan estimere stories, skrive metoder, refaktorere koden, bruge designmønstre etc. Boehm har vurderet, at jo mere erfaring en udvikler har, jo bedre er han eller hun positioneret for, at arbejde agilt. En Level 1A udvikler anses ikke i den oprindelige model som værende specielt nyttig, og derfor må vi placere os selv ganske langt ude på akse.

### 3.2.6 Size

Teamet består af fire medlemmer, det kan ikke siges at være mange. Modellen specificerer, at jo flere medlemmer et team består af, jo længere ude på akse, og i retning af en plandrevet model, skal man vælge. Her er vi altså i en klar position til at vælge en agil

løsning.

### 3.2.7 Valget



Figur 4 - Projektmap for LevelUP

Umiddelbart giver det ovenstående et lidt mudret indtryk af hvilken type udviklingsmetode vi skulle vælge. Dog er alle akser ikke født lige, og der kan være gode grunde til, at vi skulle vægte dem lidt forskelligt. For det første mener vi ikke, at vores mangel på erfaring bør afholde os fra, at vælge en agil udviklingsmetode. Det er klart, at jo mere kompetent en udvikler er, jo mere overskud vil han eller hun have i forhold til, at holde de ekstra bolde i luften, som en agil udviklingsmetode kræver. Men i vores tilfælde anser vi vores team som en iværksættervirksomhed, der udvikler eget produkt. Derfor kan vi ikke have vægten placeret på, at skulle skrive stakkevis af dokumentation, som jo ellers ville være det som gav fordelene til de mere uerfarne udviklere.

Omvendt er størrelsen på teamet en rigtig god grund til at vælge en agil løsning. Ikke mindst når man tager manglen på erfaring i udviklerteamet i betragtning. Jo flere medlemmer der er, jo flere bolde kommer der i luften, og jo sværere kan det det dermed blive, at bevare overblikket uden omfattende dokumentation. På den anden side vil det tage megen tid væk fra udviklingsarbejdet hvis denne dokumentation skulle udformes.

Det ville til gengæld trække i retningen af plandrevet for os, at størstedelen af gruppen overordnet set trives bedst med order. Her er der dog en væsentlig påvirkning udefra der afgør, at denne akse ingen betydning har. Såfremt det ikke var en forudsætning for projektet, at arbejde agilt, ville dette være en vægtig grund til at overveje en plandreven udviklingsmetode.

Når det kommer til Criticality, så er projektet af en natur hvor kritisk fejl maksimalt kan gøre skade på virksomhedens overlevelsesmuligheder. Det er selvfølgelig skidt for firmaet, men i en større kontekst er det i den milde ende, og derfor er det i den sammenhæng velegnet til at blive udviklet agilt.

Dynamism har ikke den store indflydelse for vores projekt i dette tilfælde, men den akse vi kalder Technology udspringer herfra. Det kan godt ske, at der ikke kommer de store ændringer til selve produktet. Men der er absolut grund til, at tro der kommer ændringer til den tekniske udførsel af projektet. Mest på grund af de nye teknologier som vi anvender til udviklingen, men også som følge af den manglende erfaring med, at arbejde uden tekniske dokumenter. Samlet set anser vi dette for at være den absolut væsentligste grund til at vælge en agil metode til projektet.

### 3.2.8 Risikostyring

For effektivt at kunne håndtere risiko i vores projekt, har vi i sprint 0 udført en relativt omfattende risikoanalyse, hvor fokus har været på at finde svarene på de følgende spørgsmål:

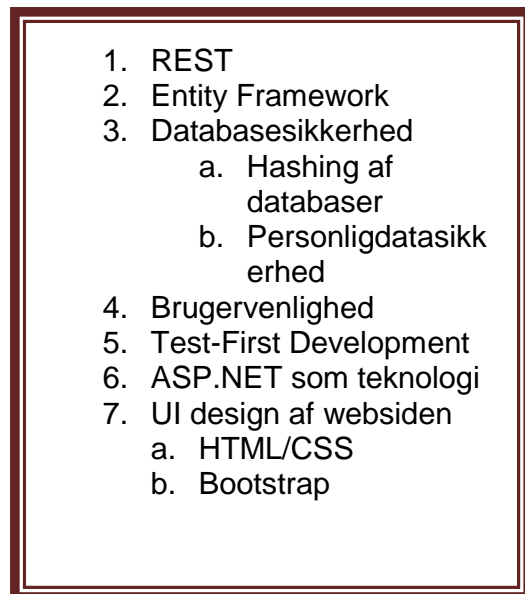
- Hvori består den mulige risiko?



- Hvad er den vurderede sandsynlighed for, at det hænder?
- Hvad er konsekvensen hvis det sker?
- Hvordan kan vi håndtere/undgå risikoen?

### 3.2.9 Risikoanalyse for LevelUp

For at kunne opnå det, har vi kigget på de krav som product owner har ridset op for os, og de teknologier som vi har udledt af de krav, og formuleret en risikoliste:

- 
1. REST
  2. Entity Framework
  3. Databasesikkerhed
    - a. Hashing af databaser
    - b. Personligdatasikkerhed
  4. Brugervenlighed
  5. Test-First Development
  6. ASP.NET som teknologi
  7. UI design af websiden
    - a. HTML/CSS
    - b. Bootstrap

Figur 5 - Udsnit af risikolisten. Se bilagssektionen for den fulde liste

Ud fra denne, har vi så lavet en tabel med en konkret beskrivelse af den risiko vi forudser, hvor sandsynligt det er for at ske, hvilken effekt (relativt til de andre risici) det har for projektet, samt en relativ skala for hvor højt risikoen skal prioriteres.

ID	Beskrivelse	Sandsynlighed	Effekt	Ranking
1	Problemer med kodning og opsætning af REST service	9	10	95
2	Der kan opstå problemer med at bruge Entity frameworket, pga. manglende erfaring	5	9	75
3	Manglende erfaring med at implementere sikkerhed i forbindelse med brugerdata	9	3	35
...	...	...	...	...

Figur 6 - Udsnit af risikomodel - Se bilagssektionen for den fulde tabel

Vi forventer at størstedelen af vores risikomomenter vil opstå som følge af manglende erfaring med de teknologier vi skal arbejde med. For at kunne stå imod når, eller hvis, disse problemer opstår har vi opstillet tabellen i figur 3, hvor hver risiko er prioriteret efter dens

ranking, med en løsningsmodel.

Som det fremgår har vi til hensigt, at spike os ud af så meget som vi kan i sprint 0.

Risiko Beskrivelse	Ranking	Mitigation
1. Problemer med Kodning og opsætning af Webservice	95	Kode simpel hello world prototype af REST service
2. Der kan opstå problemer med at bruge Entity frameworket, pga. manglende erfaring	75	Kode simpel prototype med oprettelse og opdatering af database med code-First
6. ASP.NET er en model der ikke er kendt af udviklergruppen	70	Kode simpel hello world prototype i ASP.NET
...	...	...

Figur 7 - Udsnit af mitigation tabel - Se bilagssektionen for den fulde tabel

### 3.2.10 Arkitektur

I dette sprint er der sidst blevet defineret arkitektur. I vores tilfælde ønsker product owner et produkt der kan holde dynamisk og persistent data, og som kan tilgås via en hjemmeside. På længere sigt skal systemet kunne udvides til, at omfatte apps til android, iOS og Windows Phone. Ud fra de kriterier kan vi så som udviklere skimte omridset af en arkitektur, der har en database, og et interface som kan tilgås fra disse platforme.

Almindelig sund fornuft siger os, at vi bør sigte efter en lagdelt (omend, det ikke fra starten er præcist besluttet hvor mange lag vi taler om) arkitektur.

For product owner er udviklingstiden en væsentlig faktor, derfor anser vi det som fornuftigt, at lægge vægt på en slank klient, og en fed backend. Det er ikke en beslutning uden ulemper, men det betyder, at jo mere logik der ligger i backend, jo mindre skal skrives igen når nye klienter skal udvikles.

Med disse overvejelser i hånden, og formaningen fra XP paradigmet om hyppig refaktorering, har vi den mængde prædefinerede arkitektur fra start som vi har til hensigt at have.

## 4 Sprint 1

### 4.1 Planlægning

#### 4.1.1 Velocity

Som tidligere nævnt i sprint 0 afsnittet er vores productbacklog prioriteret i samarbejde med vores product owner i sprint 0. For at finde ud af hvilken velocity vi skulle arbejde ud fra i det første sprint skulle en middel sværhedsgrad sættes til 5 ifølge reglerne fra planning poker. Her valgte vi user storien "tilføj log" som en middelopgave og diskuterede frem og tilbage om hvor lang tid den ville tage. Vi nåede frem til et konsensus af, at det ville tage 4 mandedage at færdiggøre den user story. Herefter divideres de 5 middel story points med de 4 mandedage som giver en pointer om hvor mange story points man kan nå pr. dag. Så det vil sige 1,25 story points pr. dag i sprint 1. Vi er 4 mand i gruppen og derfor ganges 4 med 5 dage som sprintet består af som giver 20 mandedage som så ganges med de 1,25 story points som kan opnås pr. dag hvilket lander os på 25 story points i alt for sprint 1. For at korrigere for par programmering har vi så divideret de 25 med 2. Dette giver os så 12,5 story points som skal udføres for sprint 1.

#### 4.1.2 Planning poker

Med vores middel user story "tilføj log" som er estimeret til 5 story points var det nu på tide at gå i gang med planning poker for at finde ud hvor mange story points de andre user stories skulle have. Inden vi gik i gang lavede vi lige en hurtig estimering af hvor mange user stories vi kunne nå, for at finde ud af hvor mange user stories vi skulle lave planning poker for. Derfor lavede vi estimering på de første 4 user stories i productbackloggen. Efter vild diskussion endte vi med at give de første 4 user stories følgende point:

- Tilføj log – 5 point
- Progress tab – 3 point
- Beregning af xp – 8 point
- Tildel achievements – 8 point

#### 4.1.3 Fra productbacklog til sprintbacklog

Nu skulle der føres opgaver fra productbackloggen over til sprintbackloggen. De højest prioriterede skal selvfølgelig tages ind først. Dette medførte dog at der var enten alt for lidt point for sprintet eller alt for mange point. De første 3 stories giver 16 point og de første to giver 8 point. Derfor snakkede vi om at tage en lavere prioriteret user story ind i stedet for, så det passer bedre med pointene for sprintet. Efter at have diskuteret hvilke user stories som kunne tages ind, blev vi enige om "opret bruger". Mens vi diskuterede snakkede vi også om at den ville give god mening at lave som en af de første, da brugeren skal bruges hele vejen igennem systemet. Men før at vi endegyldigt kunne tage beslutningen indragede vi vores product owner i problemstillingen. Product owner synes det gav mening at få flyttet "opret bruger" ind i det første sprint, så derfor er de følgende user stories valgt til vores sprintbacklog i sprint 1:

- Tilføj log – 5 point
- Progress tab – 3 point
- Opret bruger 3 point

Sammenlagt giver de 3 stories 11 point, hvilket passer fint med de 12,5 point vi har som velocity for sprintet. Progress tab userstory blev flyttet til sprint 2, da den ikke blev brændt

## 4.2 Review

Målene for sprint 1 var at få færdiggjort "Tilføj log", "Progress tab" og "Opret bruger". Som det kan ses på burndown chartet se bilag ? er det kun "opret bruger" og "tilføj log" som er blevet lavet færdig i sprintet. Progress tab skal derfor overføres til sprint 2.

Den første dag i sprintet blev der ikke brændt nogle SP dog er dette ikke alarmerende da den første dag i sprintet skal bruges på planlægning af det pågældende sprint. Vores Rest-service spike fra sprint 0 havde vi ikke fået færdig. Det var vigtigt for product owner at Rest-servicen kom op at køre så at programmet kunne konsumeres af flere forskellige klienter. Så derfor havde vi en udvikler til at arbejde på spiken den 2 dag i sprintet, hvor den også blev færdiggjort. Onsdag fik vi så lavet de 2 user stories "Tilføj log" og "Opret bruger" som vi så brændte torsdag morgen til det daglige scrum møde. Den sidste dag i sprintet nåede vi desværre ikke at få lavet user storien "progress tab" færdig, da der som nævnt tidligere var brugt tid på at lave en spike færdig fra sprint 0. Derudover skulle udviklerne også lige vænne sig til at arbejde med test-first og unittesting i visual studio. Disse faktorer var med til at der kun blev brændt 8 SP for sprintet.

## 4.3 Retrospective

### 4.3.1 Hvad var godt ?

På den positive side kan det siges at selvom vi havde 2 dage med en fraværende udvikler og var nød til at arbejde på en spike som ikke var lavet færdig og som ikke var planlagt i sprintet, er det alligevel lykkedes os at få brændt 8 sp. Med disse faktorer taget i betragtning ser vi vores estimering af sp på vores user stories som okay.

Der er delvist arbejdet med parprogrammering og det har fungeret fint i det omfang vi brugte det. Der er dog forskel på udviklere og ikke alle er lige glade for at skulle sidde 2 ved en skærm. Samtidig er der også en stor andel i gruppen som lærer bedst ved at gøre tingene selv og prøve sig frem. Der kan man sige, at parprogrammering halter lidt på det område. I øvrigt sidder vi ved siden af hinanden og kan hele tiden se hvad hver især laver og hver enkelt udvikler hjælper til og kommer med ideer.

### 4.3.2 Hvad var ikke så godt ?

Det var ikke så godt at vi havde en spike som ikke var færdig i sprint 0 som gjorde at der blev brugt tid som der ikke var planlagt i sprintet. Men på den anden side synes vi og product owner, at det var en vigtig feature i systemet.

Dog er vi tilfredse med, at have fået lavet spiken færdig og samtidig at have fået brændt størstedelen af sprintets SP.

### 4.3.3 Hvad kunne vi gøre bedre til næste sprint?

Fordi vores burndown er vandret de første 3 dage af sprintet kan det være svært at se hvor meget vi har lavet. Derfor har vi snakket om at implementere en burndown for mandetimer, sådan at vi bedre kan tracke de timer vi bruger på de forskellige opgaver og userstories. Derudover skal vi tilsikre, at vi har taget højde for alt hvad vi laver. Det er en essentiel del af Scrum, at man arbejder på det som er aftalt ved planningen, og intet andet. I dette tilfælde skulle vi have sat tid af til den spike.

## 5 Sprint 2

### 5.1 Introduktion

I sprint 2 blev der udviklet en god del på systemet. Vi fik skrevet vores første ordentlige tests, som tester på data i databasen. Vi fandt ud af, at der i forbindelse med at kode en task, nemt kan opstå uforudsete problemer, der også tager tid, og at man så må beslutte hvordan denne ekstra forbrugte tid skal håndteres og registreres i arbejdsprocessen.

I dette sprint var der primært fokus på kodning og udvikling. Mange af de måder vi havde arbejdet på i sprint 1 fortsatte vi med i dette sprint, da der via vores sprint retrospective for sprint 1 ikke var indikation for at lave store ændringer i vores arbejdsprocess. Vi havde ved sprintets start stadig en user story fra sprint 1 som ikke var færdig. Derfor var det første vi arbejdede på i dette sprint.

Vi begyndte i løbet Sprint 1 at tale om, om det kunne betale sig, udover at brænde story points, at prøve at brænde timer på et særskilt burndown chart, og vi havde også overvejelser om, om vi skulle prøve det i sprint 2. Vi besluttede at prøve det i sprint 2, og lavede derfor et burndown chart for timer, i håb om at vi kunne spore mere nøjagtigt, hvad vi fik lavet færdigt hvornår.

### 5.2 Planlægning

Ved sprintets start havde vi sprint planning meeting. Her flyttede vi, i dialog med product owner, de user stories der havde højest prioritet fra product backloggen over i sprint backlog. Vi stod efter dette med en omprioriteret ny version af vores product backlog. Efter dette lavede vi planning poker for, at estimere alle user stories for sprint 2 og nedbryde dem i tasks. Det er vigtigt at nævne, at vi ikke justerede vores burndown chart, selvom vi havde en ufærdig user stories fra sprint 1 med i sprint 2. Dog justerede vi i denne sprint planlægning vores velocity med udgangspunkt i, at vi ikke havde nået alle user stories i sprint 1. Derfor justerede vi vores velocity ned fra 12.5 til 10.0.

I planlægning af sprint 2 havde vi ikke taget højde for, at det meste af mandag vil gå til andre ting end udvikling, dermed mister vi omtrent en dag. Mandag var der både sprint review for hele klassen samt sprint retrospective internt i gruppen. Derudover var der sprint planning meeting for sprint 2. Da vi lavede burndown chart for sprint 2 påførte vi 5 dage til udvikling, og dette resulterede i, at vi havde sat en dag på til udvikling, som vi reelt ikke havde, fordi den var tildelt planlægning af sprintet. Dette betyder i praksis, at det bliver svært, at nå det arbejde der er estimeret for sprintet. Vi nåede heller ikke, at brænde alle de story points vi skulle i sprint 2. Dette gælder begge vores burn down charts.

### 5.3 Retrospective

I sprint 2 blev der hver morgen afholdt daily scrum klokken 9:00. Scrum møderne fungerede efter hensigten, og blev en hjælp til at holde et fælles overblik. Dels over hvad der blev arbejdet på, og dels hvad problemer, hver udvikler stod med, indenfor enkelte tasks. Dette var en stor fordel, da det gav alle i gruppen indsigt i, hvad der kunne være af potentielle problemer i dele af systemet, man ikke selv direkte arbejdede på. På denne måde kunne morgenmøderne bidrage til, at alle i gruppen opnår en mere solid forståelse, af det system man er sammen om at udvikle på. Som udgangspunkt havde alle altid en ny task med fra morgen mødet. Dog med undtagelse af, at man var i gang med en nuværende task som endnu ikke var færdig, til at komme i to verify. Og i praksis er det også svært at ramme så man bliver færdig med en task ved gårsdagens afslutning, så man er klar til at tage en ny ved hver daily scrum. Tasks kan også godt være estimerede til at tage mere end en arbejdsdag.

Vi talte i løbet af sprintet om, hvorvidt vi skulle prøve, at lave en form for rotation i arbejdet, så alle prøvede at arbejde indfor alle forskellige områder af projektet. Dette var for at undgå, at vi hver især blev for domænespecifikke med hver vores viden om vores pågældende område. Dette ville også have indflydelse på, hvorvidt alle vil være i stand til at ændre på kode alle steder i systemet, jævnfør den af de 12 XP praktikker omkring 'fellow code ownership'. Vi besluttede ikke noget specifikt for dette sprint omkring det, men begyndte i større grad end tidligere at arbejde således, at alle arbejdede på flere forskellige slags ting i systemet end tidligere. Dette vil antageligt også være hensigtsmæssigt, da der i senere sprints vil være et tiltagende behov for, at alle udviklere kan begå sig i alle hjørner af systemet.

Angående extreme programming-praktikker anvendte vi ikke alle de 12 eksisterende i dette sprint, men derimod dem vi på daværende tidspunkt synes der gav mening. Vi besluttede, at undlade at anvende pair programming. Det fungerede i princippet fint, da det gav alle 4 i gruppen mulighed for at sidde med individuel kodning. Dog påvirkede det vores arbejdsprocess. Som konsekvens var det nu en mere manuel udfordring, at få delt kodeviden og opnå fælles forståelse. En anden konsekvens er, at det nemt kan ske, at der i denne situation bliver arbejdet på flere user stories, end hvis man sidder i par. Dette betyder, at der arbejdes parallelt på flere stories, end man ellers ville. Og dette kan afspejle sig i, at man ikke får brændt så mange story points per løbende dag på burndown chartet, som man ellers hvis man havde arbejdet på få user stories ad gangen. Ovennævnte arbejdsgang vil resultere i, at flere user stories der er blevet arbejdet parallelt på, ofte vil være færdige på samme tid, og derfor kan indgå i den samme brænding. Derfor ses der tendens til store knæk på vores burn down sidst i sprintet.

Sprint 2 bar præg af, at vi undervejs i sprintet opdagede, at vi manglede tasks på flere user stories. Vi indså i dette sprint, at man kan komme til at arbejde på, eller blive tvunget til at arbejde på noget nyt der skal løses, før man kan komme videre med den egentlige task man

er igang med. Vi valgte i denne situation, at lave nye tasks, til de ekstra opgaver der skulle bruges tid på, eller som vi havde brugt tid på.

Dette er vigtigt, da tiden skal registreres til den task man arbejder på, og for at undgå at tiden skrider på en aktuel task, fordi man lægger andre opgaver ind under den. Dette har også noget at gøre med erfaring i, at kunne se hvad der reelt ligger i en given task. En anden ting vi opdagede i dette sprint var, hvor vigtigt det er at kommunikere undervejs i arbejdsprocessen, for at sikre en fælles forståelse af den ønskede funktionalitet. Et eksempel på dette var, at på vores updateuser profile funktionalitet. hvor systembruger kan opdatere oplysninger på sin brugerprofil, passede de input felter der er i brugerfladen ikke med de attributter der er på en user i databasen. Dette er relevant fordi det tager tid at lave om, og sandsynligvis vil gøre det sværere at holde sig indenfor taskens estimerede tid.

I dette sprint fandt også vi ud af, at det var praktisk at notere numre på alle tasks, der indikerer hvilken user story de hører til. Dette var en hjælp da vi i kraft af undlade at pair programme, ofte arbejdede på to eller flere user stories ad gangen. Når vi estimerede user stories ved sprint planning meeting og nedbrød dem i tasks, estimerede vi også timer til hver task. Efter hånden som vi arbejdede os gennem tasks, noterede vi også forbrugt tid på hver task. Dette var for at blive bedre til, at overskue hvor lang tid en given task vil tage.

En gennemgående del af dette sprint var, at lave test-first. De tests der blev implementeret i dette sprint, adskiller sig tidligere tests ved, at der ikke længere testes på simpel data, men på data fra databasen som hører til systemet. Dette blev lavet tests til stort set alle user stories. Vi fandt dog ud af, at hver gang der var blevet opdateret noget et sted i systemet, påvirkede det i mange tilfælde vores tests, så de tests der hidtil havde fungeret, ikke fungerede længere. Dette resulterede i, at vi var nødt til at omskrive vores tests, så de passede til de ændringer der var lavet. Man må antage, at dette medførte overflødig arbejde i et eller andet omfang, der ikke er dokumenteret i vores timeregneskab. Dette kunne undgås ved eventuelt at vente med, at skrive testen indtil man er sikker på, hvad det er der skal testes for. Men igen kan det være en del af agil udvikling, at der skal foretages ændringer der kræver nye test scenarier.

Under sprint 2 opdagede vi, at når vi udviklede og blev færdige med user stories, kom vi til at stå med enkeltstående søjler af funktionalitet. Vi udviklede søjler der virker som enkeltstående elementer, og der opstår her en udfordring omkring, at få disse adskilte spor af funktionalitet koblet sammen og integreret med hinanden. Dette ses første gang i dette sprint. Dette kommer af, at vi ikke fra starten af, og i forbindelse med sprint planning meeting, havde gennemskuet om en given funktionalitet skal ligge det ene eller det andet sted. Et eksempel på dette problem og løsningen af det, var da vi skulle implementere den funktionalitet, der skulle tjekke om en systembrugers experience points skulle opdateres, efter at han havde indtastet en træningssession. Der var skrevet user story til hele denne funktionalitet, som om den var en søjle for sig selv. Men vi endte med at implementere denne funktionalitet, på den søjle af funktionalitet der allerede eksisterede for user, og det



giver god mening, da det er for en specifik user, at experience points skal udregnes, ligesom experience points også er en attribut der ligger på en user i systemet.

Det kan ses på vores burndown for sprint 2, at der ikke er brændt noget de første par dage. Dette korresponderer fint med der dels var en user story vi allerede var bagud med fra sprint 1, og at vi dels havde kalkuleret med at have hele mandag, altså 4,5 dage. I virkeligheden havde vi kun 3.5 dage, da hele mandag gik til sprint review, sprint retrospective og sprint planning. Dette kombineret med, at vi ofte arbejdede på mange user stories parallelt, forklarer at vi først får brændt story points på vores burndown sidst på ugen. Angående tests kan man eventuelt optimere workflowet, så man kommer til, at skulle skrive de samme tests færrest muligt antal gange.

## 6 Sprint 3

### 6.1 Planlægning

Dette sidste sprint i projektperioden er en dag kortere end de forgående. Derfor har vi blot afsat 12 story points til, at dække sprintet. Dermed skulle vi gerne holde os til vores ”37”-timers uge.

Igen i dette sprint har vi valgt at undlade parprogrammeringen. En af udfordringerne i det sidste sprint var, at det kollektive kodeejerskab blev sat lidt i baggrunden. Alle kan naturligvis rette i koden, uden at skulle spørge nogen om tilladelse, men det har typisk været de samme gruppemedlemmer der har stået for den samme type opgave, igennem hele forløbet. I dette sprint har vi målrettet besluttet, at ryste posen, så vi på den måde kunne sprede kompetencerne. I øvrigt har vi en hensigtserklæring i gruppen om, at oppe kommunikationsniveauet, så vi hurtigere kan overkomme eventuelle hindringer.

I den forbindelse har vi tidligere haft diskuteret hvorvidt vi burde anvende Kanban-metoden til at styre workflowet på vores tasks. Det har tidligere været en kilde til ærgrelse, at opgaver har trukket ud, hvor det ikke har været nødvendigt. Det ville vi i højere grad være istand til at gennemskue hvis vi også implementerede et Kanban-board. Det havde vi valgt at gøre hvis sprintet havde haft en normal længde. Men over en så kort periode vurderer vi, at processen med at implementere denne model, ville overskygge den værdi som vi kunne få ud af den. I øvrigt er Kanban ikke at betragte som en magisk løsning. Pga. opgavens kompleksitet vil vi også fremadrettet rende ind i opgaver der går i hårdknode. Kanbans opgave ville blot være at identificere de hårdknuder for os, og måske hurtigere end vi ellers ville have set dem.

Som i de tidligere sprints, har vi fokus på kundeinvolvering, dels fordi det er en kernet del af udviklingsmetoden, men især fordi alle erfaringer peger på, at en hurtig forventningsjustering i fællesskab med kunden, kan medvirke til afhjælpe et ellers katestrofalt skred i tidsplanen.

Sprintbackloggen:

- Sikkerhed på brugernavn og password (2 SP)
- Tildel Achievements (3 SP)
- Administrere brugerprofil privatindstillinger (5 SP)
- Login skærm, find bruger (1 SP)

### 6.2 Review

Det stod klart på andendagen af sprintet, at vi ikke ville kunne nå at lave den samlede ”administrer brugerprofil”-story. Dette skyldes at opgavens kompleksitet var større end antaget. I stedet blev vi, sammen med product owner, enige om, at splitte den op i to user

stories, så vi til dette sprint kun skulle færdiggøre "ændre kodeord", mens den resterende del af den originale userstory blev fremskudt til et teoretisk 4. sprint. Dog kunne vi demo det meste af det vi havde planlagt på. Hvad angår story points, så var den originale story på 5. Baseret på de ting som er lavet i systemet, UI, databasefelter, controller-metoder. Så har gruppen vurderet at 3 story points er brugt, hvorfor den uløste opgave er delegeret videre med en værdi på 2 story points.

De resterende opgaver fremviste vi på prototype niveau, hvilket vil sige, at vi har implementeret en simpel hashing af brugerens password, hvilket er nok til at opfylde accepttesten på den user story. En mindre fejl i koden gjorde, at vi var nødt til at vise effekten af "ændre kodeord" i databasen i stedet for klienten.

"Tildel Achievements" volder nogle problemer i klienten med visningen. Men da disse problemer er direkte forbundet til den user story som vi splittede op, så er det product owners overbevisning, at accepttesten er bestået ved visning af oprettelse i databasen.

Alt i alt har vi brændt 8 ud af 10 point i dette sprint. De 2 resterende tilhører den opgave der blev fremskudt

### 6.3 Retrospective

Selv om det i sig selv var et irritationsmoment, at vi var nødsaget til at splitte en story, så er vi dog ret godt tilfredse med, at have identificeret problemet hurtigt. Det gik både hurtigere og nemmere end i nogen af de tidligere tilfælde.

Vi synes også estimeringen af velocity var god denne gang, med kun en opdelt task og ingen uforudsete spikes. Grunden skal findes i, at vi var meget bedre til, at arbejde på flere parallelle tasks. Vores fortsæt for sprintet angående en bedre kommunikation blev holdt, og derfor var vi istand til at nå opgaverne.

Samlet viser både vores time-burndown, og SP-burndown, at dette var det sprint hvor vi nåede mest af det vi havde estimeret.

I øvrigt har dette været sprintet hvor vores enfant terrible; REST-servicen, ikke har givet anledning til hverken spikes, eller alvorlige bugs.

Det gik mindre godt med, at estimere hvilke tasks der hører med til de forskellige user stories, og det sker for tit, at vi er nødt til, at skrive nye task-sedler under sprintet. Nogle gange fjerner vi også en task der ikke giver nogen mening når man får den påbegyndt. Når vi stadig kan være nogenlunde på målet for sprintet, så skyldes det, at vi specifikt har planlagt vores velocity efter det.

Der har også været en del ændringer på eksisterende metoder, når vi tilføjer ny funktionalitet, hvilket har forårsaget en slem kattepine både i forbindelse med den opsplittede story, og i forhold til vores Achievement-story. Fordi vi religiøst følger Scrum-princippet om at alle user stories er uafhængige af hinanden, og derfor kan laves i præcis

den rækkefølge product owner fastlægger, så er vi ofte nødt til at hardcode værdier ind i metoder, da den story som skulle forsyne dataen ikke er lavet endnu. Samtidigt har vi ikke lavet nogle af de klassiske UP designdokumenter, og derfor ændrer vi ret ofte i vores design. Det betyder nogen gange at ting bliver glemt, og det skaber problemer. Vores Unit Tests er ikke nok til at afdække alle de problemer som dette medfører, da mange af dem kun er synlige fra klienten.

Vi kunne enten have en slags user story der kun var til for at knytte de her vertikale søjler af materiale sammen, men det ville være et brud på Scrum, da en sådan jo af gode grunde ikke ville kunne stå alene. Alternativt kunne vi planlægge at sætte tid af til det i hver story, som en latent task. Det ville nok være det bedste.

## 7 Overordnet retrospective for hele forløbet (Perspektivering)

Efter det sidste retrospective blev afholdt i det sidste sprint afholdt vi et overordnet retrospektive som dækker hele forløbet. Dette er gjort for, at få et generelt overblik over hvad der gik godt, hvad der ikke gik så godt og hvad der kunne gøres bedre hvis der skulle udføres eventuelle fremtidige sprints.

### 7.1 Hvad gik godt?

- Der var en god fremgang i forhold til user stories og tasks
- Der var en udmærket opdeling af user stories
- Det er med succes lykkedes os at gennemføre et sprint, hvor ukendt teknologi skulle udvikles

### 7.2 Hvad gik ikke så godt?

- Opdeling af userstories i tasks
  - Der blev oprettet rigtig mange ekstra tasks under udførelsen af en user story
- Vurdering og registrering af tid forbrugt har haltet
- Vi var ikke gode til at forudse spikes
- Manglende erfaring med testfirst har kostet tid og været konstant kilde til irritation.

### 7.3 Generelle forbedringer til hele forløbet

- Lave nogle få design dokumenter
  - Vi har snakket om at det kunne være en god ide at hvis vi havde haft for eksempel domænemodel fra starten af, eller havde lavet en generel beskrivelse af alle klasser i systemet og de attributer som de skal indeholde. der har været mange opdateringer på vores modelklasser hvilket har resulteret i mange fejl ved refaktorering, som kunne have været undgået hvis vi havde beskrevet klasserne fra start af.
- Håndtere sammenkoblingen af forskellige user stories

Vi fandt ud af, at når man færdiggør user stories uafhængigt af hinanden giver det huller i form af refaktorering og funktionalitet der skal bindes sammen, hvilket tager tid. Den tid har vi ikke regnet med i vores user stories.

## 8 Konklusion

I dette projekt var fokus lagt på udviklingsprocessen frem for softwareudviklingen, og målet med dette forløb var at anvende de værktøjer og metoder, som er blevet undervist i forbindelse med faget Systemudvikling.

Projektet, som strakte sig over 4 iterationer, bundede i XP og Scrum, hvor begge var taget stærkt i brug. Dog blev metoderne ikke brugt til komplement af forskellige årsager.

Nogle dele blev specifikt valgt fra, andre var besværlige at udføre i uddannelsesøjemed. I forhold til XP blev der lagt fokus på at følge de 12 praktikker, hvor størstedelen blev fulgt, dog ikke fejlfrit eller med hængepartier. Eksempelvis var der delte meninger om at bruge parprogrammering, hvilket resulterede i at afskaffe det en iteration. Ydermere blev planning poker brugt grundigt, men bar præg af manglende erfaring i at estimere arbejdsopgaver. Alt i alt var brugen af XP vellykket, dog med visse mangler.

Samtidig blev der gjort brug af praksisser fra Scrum, såsom daglig Scrum-møde, hvor gruppen holdte hinanden opdateret. Denne funktion fungerede rigtigt godt for gruppen, da der blev sørget for alle havde arbejdsopgaver. Endvidere blev der gjort stort brug af burndown charts. Dog var der blandede følelser omkring udbyttet af disse, hvor det senere i forløbet blev besluttet at tilføje et chart over timeforbrug for at give et mere præcist billede af fremgangen. Dette lykkedes dog kun delvist, da estimering af de enkelte task til den overordnede story var meget upræcis, hvilket betød chartet ikke var en præcis gengivelse af virkeligheden.

Softwareudviklingen var relativt effektivt, dog ikke uden problemer og spikes undervejs. Gruppens brug af stories betød funktionalitet hurtigt blev udviklet, da der ikke skulle tages højde for fremtidige problemer eller funktioner. Samtidig havde sprint 0 været til enorm hjælp, da den gav mulighed for at sætte sig ind i den nye teknologi, som skulle bruges igennem projektet. Her menes der i høj grad REST-services, som selv efter sprint 0 har krævet mange ressourcer at få sat op. Derudover har der været diverse problemer med Entity Frameworket, versionsstyring og andre småting, som dog er forventet i projektarbejde. Selvom produktet naturligvis er langt fra færdigt, så er gruppen tilfreds med den fremgang, der er opnået.

Gruppen har i sidste ende fået rigeligt med erfaring i brugen af disse metoder, men det er dog mærkbart at både XP og Scrum kræver øvelse. Derudover er det enighed om, at metoderne bruges med måde på rette tid og sted.

## 9 Kildeliste

### Bøger:

Henrik Kniberg

*Scrum and XP from the Trenches*

Henrik Kniberg & Mattias Skarin

*Kanban and Scrum, making the most of both*

Mike Cohn

*User Stories Applied, For Agile Software Development*

### Hjemmesider:

Kanban applied to Scrum

<http://www.youtube.com/watch?v=0EIMxyFw9T8>

Intro to Kanban in under 5 Minutes (What is Kanban, Learn Kanban)

<http://www.youtube.com/watch?v=R8dYLBjiTUE>

Risk Management in Agile

<http://www.scrumalliance.org/community/articles/2013/2013-may/risk-management-in-agile>

Managing Risk & Quality with Scrum.wmv

<http://www.youtube.com/watch?v=zBG75t3-BYw>

The Scrum Papers, Jeff Sutherland

<http://assets.scrumfoundation.com/downloads/2/scrumpapers.pdf?1285932052>

Extreme Programming

<http://www.extremeprogramming.org/>

<http://c2.com/cgi/wiki?ExtremeProgramming>