Danmarks Tekniske Universitet

DTU

Michael Ølund, s083237

Agil udvikling i it-baserede projekter: Et studie i agile metoders egenskaber og ligheder

Afgangsprojekt, Januar 2012

DTU Informatik
Institut for Informatik og Matematisk Modellering

Agil udvikling i it-baserede projekter: Et studie i agile metoders egenskaber og ligheder
Afgangsprojekt, Januar 2012

Denne rapport er udarbejdet af:

Michael Ølund: _____

Vejleder:

Hubert Baumeister

DTU Informatik

Danmarks Tekniske Universitet

2800 Kgs. Lyngby

Denmark

reception@imm.dtu.dk

Projektperiode:	5. september 2011 – 9. januar 2012
ECTS:	20
Uddannelse:	Diplom
Retning:	Internetteknologi & Økonomi
Klasse:	Offentlig
Udgave:	1. Udgave
Løbenummer:	IMM-B.Eng-2010-90

Resumé

Baggrund: Agil software udvikling har fokus på at finde nye måder at planlægge softwareudvikling på, så der hurtigt kan leveres ny software, når der kommer ændringer fra kunden. Der findes dog ikke én agil udviklingsmetode, men mange, der hver især har forskellige karakteristika og egenskaber. I løbet af årene er der kommet nye agile metoder til og nogle er blevet revideret og ændret undervejs. Dette har skabt en jungle af agile udviklingsmetoder, der alle hævder, at kunne være med til at udvikle software, der giver værdi for kunden på en hurtig og effektiv måde. Ud fra denne jungle af agile udviklingsmetoder kan det måske være svært for en projektleder eller udvikler at overskue alle disse metoder, hvilket dermed kan besværliggøre valget af metode.

Formål: Fire agile udviklingsmetoder, er i projektet analyseret med det formål at lave en sammenligning. Derudover er der lavet en empirisk undersøgelse, med henblik på at finde ud af om man ud fra projekters parametre kan forudsige, hvornår en bestemt agil metode vil virke.

Metode: Metoden anvendt til sammenligning af de fire agile metoder, bygger på et omfattende framework til sammenligning af agile metoder. Derudover er der blevet afholdt interviews, baseret på dette framework, med fem projektledere fra forskellige danske virksomheder.

Resultater: For en sammenligning af de fire agile metoder, er det vist at der er ligheder i de undersøgte dimensioner, i flere af de agile metoder. Derudover er det vist, at den agile metode benyttet i de fem undersøgte projekter, i høj grad er blevet tilpasset med praktikker fra andre agile metoder. Der er vist at der ikke er en signifikant sammenhæng mellem de fem undersøgte projekternes parametre.

Konklusion: De fire agile udviklingsmetoder er sammenlignet ud fra fem dimensioner; proces, modellerings sprog, agilitet, brugbarhed og cross-context. Det er konkluderet at der ikke kan vises en signifikant sammenhæng mellem de fem undersøgte projekternes parametre. Derfor kan der ikke konkluderes på at den agile metode, de har benyttet, vil virke på et specifikt projekt. Der er vist at der ikke udelukkende ud fra projekters entydige parametre kan bestemmes hvornår en agil metode vil virke på et konkret projekt. Endeligt konkluderes det, at de foreslåede kvantitative værdier fra undersøgelsesmetoden, ikke vil kunne bidrage til en sammenligning mellem agile metoder.

Abstract

Background: Agile software development have focus on finding new ways of planning software development, so that software, quickly can be delivered, when changing requirements from the customer is received. However, there are not just one agile development methodology, but many, and all with different characteristics and features. During the years many new agile methodologies has been born, and some has been revised. This has created a jungle of agile methodologies, which all claim to help the software development process. From all these methodologies it might be hard for a project manager or developer to get a good overview of all these methods, which might affect the choice of method.

Objective: Four agile methodologies, has been analysed in this study with the objective to make a comparison. Additionally, there has been made an empirical study, with the objective to find out, if a set of project parameters can predict, which specific agile method will work.

Method: The method used for comparison between the four agile methods, relies on a comprehensive framework for comparison of agile methods. In addition to this, interviews based on this framework, have been made with five project managers from different Danish organizations.

Results: For a comparison of the four agile methods, there is shown similarities in the investigated dimensions, in several of the agile methods. Additionally, it is shown, that the agile methods used in the five organizations, to a large extend have been customized with elements from other agile methods. It is shown, that there are no significant connection between the five investigated projects parameters.

Conclusion: The four agile methods has been compared from five dimensions: process, modelling language, agility, usage and cross-context. It is concluded that there are no significant similarities between the five investigated projects parameters. Therefore no conclusion can be made, about the agile method they used also will be usable on a specific project. It is shown that not only the projects parameters can determine which agile method should be used on a particular project. Finally, it is concluded that the quantitative values from the studymethod, can't be used for a comparison between agile methods.

Forord

Dette projekt er udarbejdet på Institut for Informatik og Matematisk Modellering, som afslutning på diplomingeniøruddannelsen i Internetteknologi & Økonomi. Arbejdet er udført fra september 2011 til januar 2012 og dækker en arbejdsbyrde på 20 ECTS-point for forfatteren.

Der skal lyde en stor tak til min vejleder for kompetent vejledning:

- Lektor, Hubert Baumeister
  Institut for Informatik og Matematisk Modellering, DTU IMM
  Danmarks Tekniske Universitet

Michael Ølund

DTU, Januar 2012

Indholdsfortegnelse

1 Introduktion

1.1 Bagrund

Agil software udvikling anerkender, at kunden nok ikke altid kender alle detaljer og krav til et projekt fra dets start. Derfor er der fokus på at finde nye måder at planlægge softwareudvikling på, så der hurtigt kan leveres ny software, når der kommer ændringer fra kunden. Agile metoder fokuserer derfor på, at bryde arbejdet op i små iterationer, og planlægningen vil derfor kun vare et par uger eller måneder ud i fremtiden.

I 2001 blev det Agile Manifest dannet af 17 softwareudviklere, der diskuterede letvægts udviklings metoder. De kom frem til fire grundlæggende værdier i agil udvikling [i]:

- Individer og samarbejde frem for processer og værktøjer
- Velfungerende software frem for omfattende dokumentation
- Samarbejde med kunden frem for kontraktforhandling
- Håndtering af forandringer frem for fastholdelse af en plan.

Der findes dog ikke én agil udviklingsmetode, men mange, der hver især har forskellige karakteristika og egenskaber. I løbet af årene er der kommet nye agile metoder til og nogle er blevet revideret og ændret undervejs. Dette har skabt en jungle af agile udviklingsmetoder, der alle hævder, at kunne være med til at udvikle software, der giver værdi for kunden på en hurtig og effektiv måde.

Ud fra denne jungle af agile udviklingsmetoder kan det måske være svært for en projektleder eller udvikler at overskue alle disse metoder, hvilket dermed kan besværliggøre valget af metode.

1.2 Problemformulering

Projektets hovedformål er at lave en sammenligning af agile udviklingsmetoder. For begrænsningens skyld er der udvalgt 4 forskellige metoder til analysen. Derudover vil opgaven også fokusere på, om man ud fra en sammenligning kan finde ud af, hvornår en agil metode vil virke på et projekt, og hvordan man bedst muligt vælger en metode. Opgaven er målrettet mod:

- En analyse af fire agile metoders teori, med henblik på at vise sammenhænge mellem metoderne.
- En undersøgelse af agile metoder i virksomheder, med henblik på at finde ud af om man ud fra projekters parametre kan forudsige, hvornår en bestemt agil metode vil virke.

Undersøgelsen har fundet sted i fem private danske virksomheder, ved interviews med projektledere.

1.3 Overblik over rapporten

Denne rapport starter med et indledende afsnit omkring fire agile udviklingsmetoder. Det indledende afsnit er efterfulgt af et state-of-the-art afsnit, hvor de vigtigste tendenser fra den øvrige litteratur indenfor området er præsenteret. Herefter præsenteres den anvendte metode der er brugt til sammenligningen af metoderne. I kapitel 5 vil resultaterne fra analysen blive præsenteret. Endeligt diskuteres resultaterne og den anvendte metode. Opgaven afsluttes med en konklusion.
Størstedelen af litteraturen benyttet til dette projekt er skrevet på engelsk. Derfor er der i de fleste tilfælde

valgt ikke at oversætte fagudtryk for at undgå forvirring ved dårlige oversættelser. Engelske fagudtryk er markeret med kursiv.

2 Teorien bag agile udviklingsmetoder

I dette afsnit vil den grundlæggende teori til de fire agile metoder, jeg har fokuseret på, blive gennemgået. Afsnittet starter med en kort grundlæggende introduktion til de agile metoder, og derefter vil metoden blive gennemgået ift. proces, roller, praktikker og anvendelse.

2.1 Extreme Programming

Kent Beck beskriver *Extreme Programming* (XP) som værende en enkelt, effektiv, sikker, fleksibel, videnskabelig samt sjov måde at udvikle software på[ii].

XP er baseret på fem overordnede værdier[ii]:

- Kommunikation
- Enkelthed
- Feedback
- Mod
- Respekt

Kommunikation
Når et problem opstår i et projekt, kan oprindelsen næsten altid spores tilbage til, at der er nogle, der ikke har talt med hinanden. Det kan både være fra udviklernes side, hvor en ændring i designet ikke bliver kommunikeret ordentligt videre til kunden. Eller fra kundenss side der ikke har kommunikeret ændringer i et krav ordentligt videre til udviklerne.
XP prøver at afhjælpe kommunikations problemer ved at indføre en række arbejdsvaner der tvinger udviklere og kunden til at tale sammen, og det skaber værdi i sidste ende.

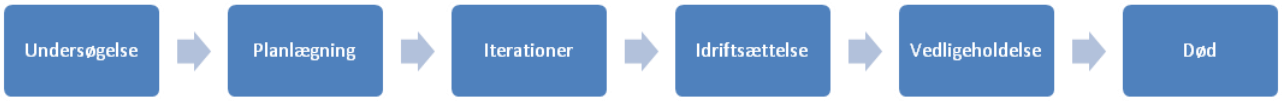
Enkelthed
I XP har man den indstilling, at det er bedre at gøre noget enkelt i dag, og så betale lidt mere for det i morgen, hvis det bliver nødvendigt at ændre det. Denne indstilling skal ses som, at man ikke starter med at lave noget meget kompleks, som måske i sidste ende slet ikke ender med at blive brugt. Derfor er det bedre at starte enkelt, og så bygge videre på det.

Feedback
Feedback fungerer helt fra små perioder på minutter til længere perioder af måneders varighed. Når udvikleren skriver *testcases* for alle de dele af koden, der vil kunne fejle, vil han få en aktuel feedback fra minut til minut på systemets aktuelle tilstand. Efter at kunden har skrevet *user stories* til en given funktionalitet, vil udvikleren estimere, hvor lang tid denne vil tage at udvikle. Dermed vil kunden få feedback på den beskrevne funktionalitet, og samtidig kan udvikleren få mere feedback fra kunden, hvis der er noget af beskrivelsen der er uklar.

Mod
Denne værdi går ud på, at man kaster sig ud i tingene – men kun hvis de tre første værdier er opfyldt. Det kan fx være at udskifte en hel dags kode, som man egentlig ikke bryder sig om, eller prøve skøre ideer af, som måske kan give værdi. Mod hænger tæt sammen med hver af de første værdier. Kommunikation kan skabe mod. Dialog med kollegaer kan skabe ideer og forslag, som så diskuteres og måske accepteres. Enkelthed kan også skabe mod. I og med systemet er enkelt, er konsekvenserne hvis noget går galt overskuelige. Til sidst vil feedback også skabe mod, i og med *testcases* hurtigt og præcist vil kunne vise, hvilken effekt ændringen i koden har på hele systemet.

Respekt
Hvis teammedlemmer er ligeglade med hinanden, og med hvad de laver, vil XP ikke virke. Hvis medlemmerne er ligeglade med et projekt, er der ikke noget, der kan redde det. Derfor er det vigtigt at have respekt for hinanden og for projektet.

2.1.1 Proces
Et XP projekt vil typisk gennemløbe 6 forskellige faser [iii], illustreret i Figur 1.



Figur 1- XP's 6 typiske faser

Undersøgelse: Kunden vil starte med at skrive de *user stories,* som de vil have med i første *release*. Hver *user story* beskriver en funktionalitet, som skal tilføjes programmet. Undersøgelsesfasen bliver af projektteamet brugt til at blive bekendt med værktøjer, teknologi og processer, de vil benytte sig af i projektets levetid. Fasen slutter først, når kunden er sikker på, der er skrevet nok *user stories*, til at der kan laves en acceptabel leverance, og når udviklerne har estimeret *user stories*.

Planlægning: Når alle *user stories* er skrevet vil der blive udarbejdet en prioritetsliste for alle *user stories*. Formålet er, at kunden prioriterer de *user stories* der skal indgå i næste iteration, og udviklerne vil udarbejde estimater på hver af de valgte *user stories*. På denne måde vil planlægningen foregå i starten af hver iteration, og sikre god dialog mellem udviklere og kunden omkring funktionaliteten af systemet.

Iterationer: Denne fase indeholder en række iterationer der leder op til den endelige release af systemet. Alle user stories vil blive brudt ned i iterationer, som hver vil have mellem én til tre ugers varighed. Det er kunden, der bestemmer hvilke *user stories,* der vælges til hvilke iterationer. Ved slutningen af hver iteration vil der køre en funktionalitets test, som er udarbejdet af kunden. Når iterationen er færdig er systemet i princippet klar til at gå i drift, men først når alle iterationer er færdige vil systemet være færdigt.

Idriftsættelse: I denne fase vil der være ekstra fokus på test samt performancemålinger af systemet, inden det kan blive leveret til kunden. I denne fase vil udviklingshastigheden blive sænket. Det betyder blot, at der vil være større fokus på at vurdere om en bestemt ændring skal komme med i leveranchen eller ej. Ideer, som bliver udskudt eller ikke kommer med i leveranchen, bliver dokumenteret for at kunne blive implementeret på et senere tidspunkt, fx i vedligeholdelsesfasen.

Vedligeholdelse: I vedligeholdelsesfasen skal projektet både levere ny funktionalitet samt holde det eksisterende system kørende. Der vil naturligt komme nogle support opgaver, som udviklerne også skal tage sig af. I og med at systemet er i drift vil udviklingen være mere forsigtig, når der foretages ændringer.

Død: Den sidste fase starter, når kunden ikke længere har flere *stories,* der skal tilføjes systemet. Dette kræver dog, at kunden er tilfreds med alle aspekter af systemet – bl.a. performance. På dette tidspunkt i projektet skal der skrives den endelige dokumentation omkring systemet, og der vil ikke længere blive lavet ændringer i design, arkitektur eller kode. Død fasen kan dog også indtræffe, hvis projektet bliver for dyrt, eller ikke leverer nogen resultater.

2.1.2 Roller

Et hvilket som helst hold fungerer bedst, hvis der er nogle klart definerede roller og ansvarsområder, så folk tager ansvar for at udfylde deres rolle. I et XP-team findes følgende roller: *Testers, interaction designers, architects, project managers, product managers, executives, technical writers, users,* og *programmers* [ii]. Dog er rollerne ikke fastlåste, og en person kan have elementer af flere roller. Fx kan en udvikler også fungere som arkitekt. De væsentligste roller er beskrevet herunder:

Programmers er selvfølgelig en væsentlig hjørnesten i XP. Det er dem, der skriver koden så enkelt og overskuelig som muligt. Derudover skriver de *test cases* for at demonstrere de vigtige egenskaber ved systemet. For at XP-projektet bliver succesfuldt er det vigtigt at udviklerne kommunikerer og koordinerer med andre teammedlemmer.

Users skriver *user stories* og de funktionelle test, og de beslutter hvornår et krav er opfyldt. Derudover er det også *useren,* der definerer prioriteterne for alle *stories* og dermed bestemmer, hvilken rækkefølge kravene bliver implementeret.

Testers hjælper *users* med at skrive de funktionelle tests. Testeren skal sørge for, at der jævnligt bliver kørt funktionelle test, og at resultaterne bliver offentliggjort i teamet.

Executives tager beslutningerne, og sørger for at teamet har de nødvendige ressourcer for at kunne fuldføre opgaven. Dette gøres i tæt samarbejde med projekt teamet, for at afklare den pågældende situation og finde frem til løsningen på potentielle problemer eller udfordringer.

2.1.3 Praktikker

For at XP fungerer er der udviklet en række regler, som hver især er med til at styrke processen.

Der er opstillet følgende primære praktikker for XP[ii]:

Sit Together går ud på at hele teamet helst skal sidde samlet i samme kontor. Dette vil styrke kommunikationen i teamet, og sørge for at udviklere hurtigt kan få afklaringer fra kunden.

Whole Team pointerer at det er vigtigt at teamet føler sig som et team. Teammedlemmer kan have forskellige kompetencer og baggrunde, så det er vigtigt at opnå følelsen af at man er ét team. Dette vil også styrke samarbejdet med forretningen.

Informative Workspace foreslår at det lokale der udvikles i, bør "udsmykkes" med informationer omkring projektet. Det kan fx være i form af *user stories* på væggen, der kan give et overblik over projektets indhold.

Energized Work anbefaler at man kun skal arbejde så mange timer som man kan være produktiv.

Pair Programming foreslår at al koden skrives med to personer siddende på en PC. Dette vil skabe en god dialog mellem udviklerne mens der programmeres, og kan være med til at sikre en bedre kvalitet af koden.

Stories er funktionalitets beskrivelser som alle kan forstå. *Stories* bliver estimeret tidligt i processen, for at et overblik over hvad en given funktionalitet vil "koste" at udvikle. *Stories* og estimeringen af dem vil skabe værdi og overblik for kunden når han skal prioritere.

Weekly Cycle foreslår at der kun planlægges en uge ud i fremtiden. I starten af en uge holdes der et møde hvor der vil være et *review* af fremdriften, og kunden vælger den næste uges *user stories* der skal implementeres. Udviklerne vil så bryde *user stories* yderligere ned i *tasks* og vælge hvem der udvikler hvilke *stories*.

Quarterly Cycle foreslår at der hver kvartal reflekteres over teamet, projektet, fremdriften og målene.

Slack anbefaler at der inkluder nogle mindre opgaver i planen, der kan blive droppet hvis man kommer efter planen. Der kan altid tilføjes flere *stories* hvis man kommer forud for planen.

Ten-Minute Build er et ideal der foreslår at hele systemet skal automatisk bygges og testen på 10 minutter.

Continuous Integration foreslår at ny kode eller ændringer i eksisterende kode integreres og testes efter et par timer.

Test-First Programming foreslår at der skrives en fejlende *test case* før der ændres på noget kode.

Incremental Design foreslår at der fokuseres på designet gennem hele processen, og ikke kun før implementeringen starter.

2.1.4 Anvendelse

XP anbefaler at metoden benyttes på mindre teams. Det er vigtigt, at det er nemt og hurtigt for udviklerne at kommunikere og koordinere, det betyder dog ikke, at XP ikke kan bruges på distribuerede teams, blot at udviklere vil være mere produktive når de sidder sammen. Virksomhedskulturen kan være en hindring for at udføre XP. Hvis en virksomhed prøver at sætte kursen fra starten af, vil de få problemer med det team der kører XP og selv vil styre projektet[ii].

2.2 Scrum

Scrum er en software udviklingsmetode udviklet af Ken Scwaber. Scrum metoden er blevet udviklet for at kunne lede system udviklings processen. Metoden definerer ikke nogen konkrete software udviklings teknikker for implementerings fasen, men koncentrerer sig udelukkende om hvordan team medlemmerne fungerer for at udvikle systemet i et konstant forandrende miljø[iv].

Grundideen bag Scrum er, at software udvikling involverer flere variable (fx krav, tid, ressourcer eller teknologi) som højst sandsynligt vil ændre sig i løbet af udviklingsprocessen. Det gør udviklingsprocessen uforudsigelig og kompleks, og vil derfor kræve en vis grad af fleksibilitet for at kunne omstille sig diverse ændringer[iv].

De grundlæggende elementer i Scrum er; teamet og deres roller, *timeboxe*, artefakter, og regler.

Et Scrum team har fokus på optimering af fleksibilitet og produktivitet. Dette bliver bl.a. gjort vha. selvorganisering, tværfaglighed, og ved iterations arbejde.

Et af de væsentligste elementer i Scrum er *Sprints*, som er iterationer af et par ugers varighed. Varigheden af et *sprint* er op til det enkelte team at definere, men ligger om regel mellem et par uger og en måned. Alle *sprints* slutter af med en funktionel udvidelse af systemet, som i princippet er klar til at blive sat i produktion.

2.2.1 Proces

Scrum processen forløber over 3 faser: *pre-game*, udvikling og *post-game* [v].

Figur 2 Scrum processens 3 faser

Pre-game fasen består af to hovedområder; planlægning og arkitektur. Planlægning består i at definere det system der skal udvikles. Alle krav, som bliver identificeret, samles i en *Product Backlog* liste, som indeholder alle de krav til systemet, der foreløbig er kendt. Kravene kan komme fra alle interessenter i projektet, såvel forretningen som udviklerne. Denne liste bliver så prioriteret, og alle *user stories* bliver estimeret. *Backlog'en* bliver gennem hele processen revideret. Der kan hele tiden tilføjes nye *stories*, der kan ændres på estimaterne, og der kan ændres på prioriteringsrækkefølgen. Efter hvert *sprint* vil den opdaterede *Backlog* bliver gennemgået af Scrum teamet, for at garantere deres accept. I denne del af *pre-game* fasen bliver selve projektteamet og andre ressourcer defineret. Arkitektur delen af *pre-game* fasen består af et *high level design* af systemet, og arkitekturen bliver planlagt baseret på de *stories,* der er i *Product Baglog'en*.

Udviklingsfasen er den agile del af Scrum processen. Her bliver systemet udviklet i de såkaldte *sprints*. Et *sprint* er en iterativ cyklus, hvor ny funktionalitet bliver udviklet eller forbedret. Et *sprints* længde er defineret af Scrum teamet, og kan vare mellem en uge og en måned. Det er derfor heller ikke fastlagt hvor

mange sprints, der er i udviklingsfasen, da det kommer an på omfanget af projektet. Alle planlagte *sprints* skal være gennemført før systemet kan gå i produktion.

Post-game fasen starter, når kunden og udviklerne er kommet til enighed om, at kravene er blevet opfyldt. Når det er sket, kan der ikke længere tilføjes mere funktionalitet. Systemet vil derfor være klar til *release*, og forberedelserne til det vil så starte. I denne fase vil systemet blive dokumenteret.

2.2.2 Roller

Scrum definerer fem forskellige roller i Scrum processen[iv]. Hver rolle er ansvarlige for forskellige aspekter af processen, og har forskellige arbejdsopgaver.

Scrum Masteren er ansvarlig for, at projektet bliver udført i overensstemmelse med Scrums værdier, praksis og regler. Derudover er *Scrum Masteren* ansvarlig for at fremdriften i projektet forløber som planlagt. *Scrum Masteren* har tæt kontakt med Scrum teamet, kunden og ledelsen under hele projektforløbet. Han fungerer som en slags coach for Scrum teamet, og vil uddanne teamet til at være mere produktive og fremstille en løsning i høj kvalitet. *Scrum Masteren* er ansvarlig for at fjerne alle forhindringer/udfordringer, der ikke direkte har noget med udviklingen at gøre. Dette er for at sikre at Scrum teamet arbejder så effektivt som overhovedet muligt.

Product Owner er overordnet ansvarlig for projektet. Derudover er han ansvarlig for vedligeholdelsen af *Backlog'en* og for at sikre, den er tilgængelig og synlig for alle. *Product Owneren* bliver valgt af *Scrum Masteren*, kunden og ledelsen af projektet. Det er ham, der har det sidste ord ved beslutninger vedr. *Backlog'en*.

Scrum teamet består af udviklere, der under hvert sprint vil omforme *Backlog'en* til en ny funktionalitet, der i princippet er klar til at blive sat i produktion. Team medlemmerne kan have forskellige kompetencer, men vil tilsammen have de færdigheder, der skal til for løse opgaven. Teamet er selvorganiserende, og bestemmer derfor selv hvordan de vil omforme *Backlog'en* til ny funktionalitet. Udover udviklingen af løsningen er teamet involveret i estimering af *stories*, udarbejdelse af *Backlog'en*, *review* af *Backlog* og anbefalinger i forhold til, hvad der kan undlades/tilføjes til projektet.
Scrum anbefaler, at den optimale størrelse på et team er 7 personer, plus eller minus 2 personer. Hvis der er færre end 5 personer, vil der ikke være interaktion imellem udviklerne, og det vil mindske produktiviteten. Derudover kan de støde på problemer i form af manglende kompetencer til at levere et produkt.
Hvis teamet består af flere end 9 medlemmer, vil det resultere i overkoordination, som vil resultere i at processen vil være kompleks at styre[iv].

Kunden deltager i opgaver relateret til *Backlog'en*. Kunden kan være med til at skrive de forskellige *stories*, og er i tæt dialog med udviklerne for at afstemme forventninger og krav til funktionalitet.

Ledelsen deltager i udarbejdelsen af målsætningen og krav. Det er bl.a. ledelsen, der udvælger *Product Owner*, overvåger fremdriften, og reducerer *Backlog'en* i samarbejde med *Scrum Masteren*.

2.2.3 Praktikker

Scrum kræver ikke nogle konkrete regler omkring selve programmeringen af systemet. Men i stedet kræver Scrum nogle ledelsesmæssige praktikker og regler. Scrum deler disse retningslinjer op efter artefakter og *timeboxe*[v]

Artefakter

Product Backlog er en liste over features, funktioner, teknologier, forbedringer, og fejlrettelser som vil være med til at udvikle det fremtidige system. I *Backlog'en* er alle *stories* defineret, baseret på nuværende viden, som skal bruges til det endelige system. Denne kan altså ses som en stor to-do liste for hele projektet. Listen vil hele tiden blive opdateret, udvidet og prioriteret, og indeholder både opgaver for udviklerne og kunden. Listen er altså dynamisk, og vil udvikle sig i takt med produktet. Alle projektets interessenter kan i princippet være med til at tilføje *stories* til *Backlog'en*.

Estimaterne for de *stories* der ligger i *Backlog'en* vil blive udarbejdet i forbindelse med *Release Planning*. Når en ny *story* bliver tilføjet *Backlog'en* vil den samtidig blive estimeret. Teamet er ansvarlige for estimaterne, og de kan revideres til enhver tid.

Release Burndown grafen angiver den forventede arbejdsindsats for den resterende del af projektet. Måleenheden for tid er som regel i *sprints*. Der er ikke nogen fast defineret måleenhed for arbejdsindsats.

Sprint Backlog'en er udgangspunktet I hvert *Sprint*. Det er en liste med udvalgte elementer fra *Product Backlog'en,* der skal implementeres i det pågældende *Sprint*. Scrum teamet, *Scrum Master og Product Owner* finder i fællesskab ud af hvilke *user stories,* der skal indgå i *Sprint Backlog'en*.

Sprint Burndown er en graf næsten magen til *Release Burndown* grafen, denne graf viser blot hvor meget arbejde, der er tilbage i pågældende sprint.

Timeboxe

Timeboxe er de forskellige møder (og iteration), der findes I Scrum processen; *Release Planning* møde, *Sprintet, Sprint Planning* mødet*, Sprint Review*, *Sprint Retrospective* og *Daily Scrum* mødet[v]

Release Planning mødet har til formål at lave en plan, samt udarbejde nogle mål, som Scrum teamet forstår, og som kan kommunikeres til resten af organisationen. Det er valgfrit om, man vil benytte sig af dette møde, men det kan vise sig at blive en udfordring som senere skal løses, hvis det ikke holdes.

Sprintet er hjertet I Scrum, og består af en iteration. Længden af iterationen er valgfri, og kan være alt lige fra en uge til en måned. Et sprint består af; *Sprint Planning* mødet, selve udviklings arbejdet, *Sprint Review* mødet samt *Sprint Retrospective* mødet.

***Sprint Planning* mødet** har til formål at planlægge næste sprint. Mødet består af to dele; første del hvor der bliver besluttet hvad der skal ske i *sprintet*, og anden del hvor teamet bliver enige om hvordan de, i løbet af *sprintet*, kan udvide produktet med ny funktionalitet.

Til ***Sprint Review* mødet** vil Scrum teamet, sammen med fx kunden eller *Product Owner*, vurdere det opnåede resultat i sprintet. På denne baggrund skal det vurderes hvad næste opgave kunne være.

Sprint Retrospective vil evaluere, hvordan seneste *sprint* forløb mht. mennesker, relationer, proces og værktøjer. Dermed kan man komme frem til, hvilke elementer der fungerede, samt hvilke områder der kan forbedres for at opnå et endnu bedre resultat.

***Daily Scrum* mødet** vil samle teamet dagligt i 15 min. Her vil hvert teammedlem forklare:

- Hvad der er gennemført siden sidste møde
- Hvad der vil blive gennemført inden næste møde
- Hvilke hindringer er der i vejen for at arbejdet kan blive gennemført

Mødet er med til at forbedre kommunikationen, afhjælper problemer tidligt i processen, en hurtigt beslutningsproces, og det vil give deltagerne et godt overblik på projektet.

Konkrete regler
Der er dog et par helt konkrete regler ved Scrum[v]:

- Formålet ved hvert *sprint* er at der bliver leveret en udvidelse af systemets funktionalitet, som i princippet er klar til at blive sat i produktion.
- Ingen kan fortælle Scrum teamet, hvordan arbejde i *sprintet* skal udføres.
- Ingen kan fortælle *Product Owner,* hvordan prioriteterne i *Backlog'en* skal være.
- Ingen kan fortælle *Scrum Masteren,* hvordan han skal styre Scrum processen.

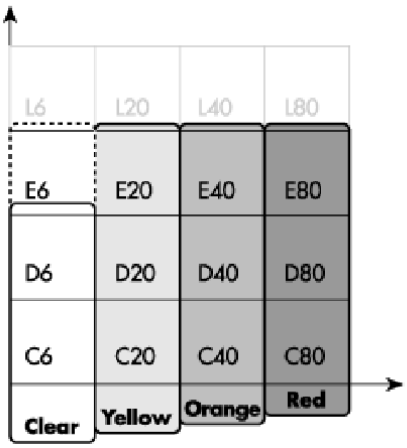
2.2.4 Anvendelse
Scrum metoden er bedst egnet for små teams med ikke flere end 10 medlemmer. Scrum foreslår, at et team har mellem 5 og 9 personer. Hvis antallet overstiger dette, kan der oprettes flere teams der arbejder på samme projekt[iv].

2.3 Crystal

Alistair Cockburn har udviklet en række af metoder, som alle hører under Crystal familien. Valget af metode afhænger af kritikalitet og størrelse på projektet[vi].

Hvert medlem af Crystal familien er markeret af en farve, som indikerer hvor omfattende metoden er. Større projekter kræver sandsynligvis mere koordination og mere omfattende metoder end et mindre projekt vil gøre.

Bokstaverne i Figur 3 indikerer kritikalitets niveauet for hver af metoderne [vi].
C står for *Comfort*, og indikerer at et kritikalitets niveau på C vil betyde, at et system nedbrud vil resultere i et tab af komfort for brugeren.
D står for *Discretionary money*, og indikerer at et kritikalitets niveau på D vil betyde, at et system nedbrud vil resultere i tab af rådighedsbeløb.
E står for *Essential money*, og indikerer at et kritikalitets niveau på E vil betyde, at et system nedbrud vil resultere i tab af essentiel finansiering.
L står for *Life*, og indikerer at et kritikalitets niveau på L vil betyde, at et system nedbrud vil resultere i et tab af liv.



Figur 3 Crystal metodernes navne fordelt efter farve. Figur taget fra [vi].

Der er nogle regler, features og værdier som går igen i alle Crystal metoderne.
Der er to regler, der går igen i alle metoderne:

- Alle metoderne bruger en trinvis udviklings cyklus men et maksimalt interval på 4 måneder, men som ideelt ligger mellem et og tre måneder.
- Teamet skal holde pre- og post-trin refleksions workshops. Crystal foreslår også at holde en midtvejs workshop[vi].

Derudover er der to basisteknikker i alle metoderne:

- *Methodology-tuning technique*: konverterer en basis metode til en accepteret metode for projektet, vha. interviews og workshops.
- Refleksions workshops

Crystal metoderne dikterer ikke nogen udviklings regler eller værktøjer, og tillader direkte tilføjelse af andre metodikker – fx Scrum og XP[vi].

2.3.1 Proces

Crystal Clear og Crystal Orange er de to medlemmer af familien der er blevet udarbejdet og brugt i praksis [vii], derfor vil jeg udelukkende fokuserer på disse to i resten af dette kapitel.

Crystal Clear er designet for små projekter med op til 6 medlemmer. Et Crystal Clear projektteam skal sidde sammen i det samme kontor for at undgå udfordringer med kommunikationen.

Crystal Orange er designet for mellemstore projekter med 10 til 40 medlemmer. I denne metode vil projektet blive delt op i flere forskellige team afhængigt at deltager antallet.

Politiske standarder
De politiske standarder er de praktikker, som skal udføres i udviklingsfasen. Crystal foreslår følgende politiske standarder[vi]:

- Regelmæssige trinvise leverancer
- Opfølgning på fremdrift med milepæle baseret på software leverancer og beslutninger
- Direkte brugerinvolvering
- Regressions test af funktionalitet
- Workshops for metode-justering

Der er dog en enkelt forskel mellem Clear og Orange her. Clear foreslår trinvise leverancer med et interval på to til tre måneder, hvor Orange foreslår, at intervallet kan forøges til 4 måneder[vi]

Work products
De *work products* som går igen i de to metoder er;

- Release sekvens
- En fælles objekt model
- Bruger manual
- Test cases
- Kode migrering

Der er dog nogle forskelle på *work products* i de to metoder.

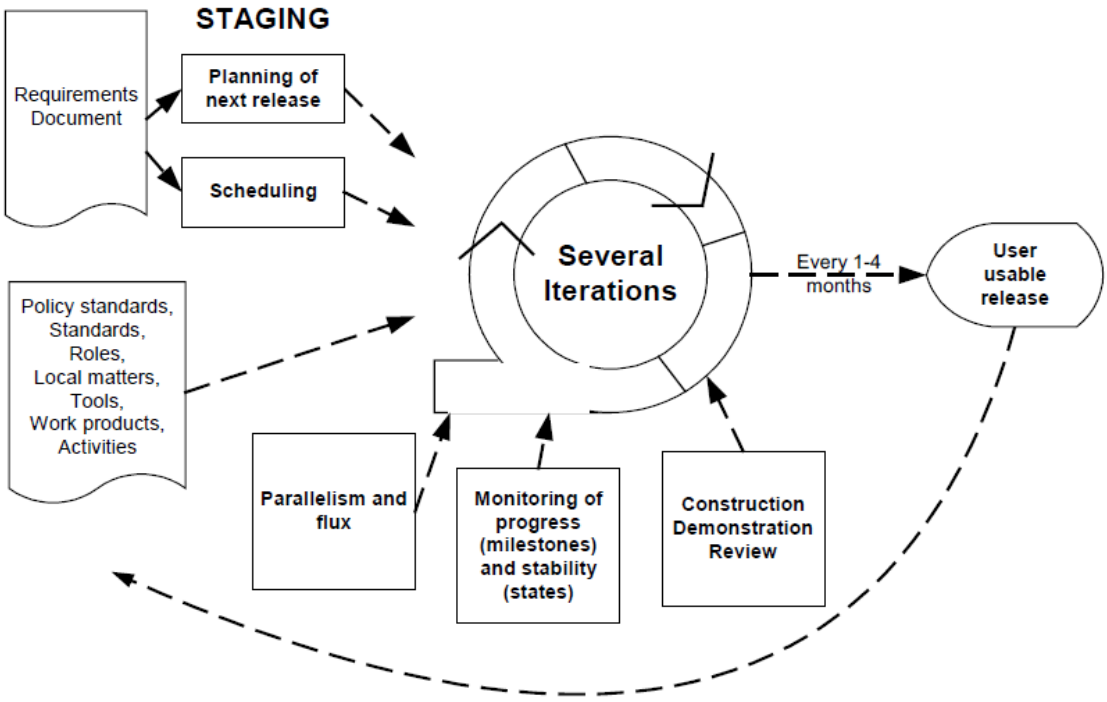
I Clear er kommenterede *use case*/kravspecifikationer et krav, mens i Orange er et decideret kravdokument obligatorisk. I Clear skal design dokumentationen blot bestå af skitser og notater, mens der i Orange kræves et *user interface* design dokument og deltaljerede specifikationer til alle teams. Derudover kræver Orange også, at der bliver udarbejdet status rapporter[vi].

Local matters
Begge metoder foreslår, at der udarbejdes:

- Templates for *work products*
- Standarder for kode og *user interface*
- Standarder for regressions test

Tools
Det vigtigste værktøjer for Crystal Clear er en *compiler*. Derudover er et væsentligt værktøj et versions- og konfigurations styrings system, samt et *whiteboard* til at visualisere præsentationer fx resultater fra et møde.
I Crystal Orange er de vigtigste værktøjer dem, der bliver brugt til versionsstyring, selve programmeringen, test, kommunikation, projekt opfølgning og performance målinger.

Et trin i processen



Figur 4 Illutration af et trin i Crystal Orange processen. Taget fra [viii]

På Figur 4 er et enkelt trin i Crystal Orange processen vist. Planlægningen tager udgangspunkt i krav dokumentet, og ender ud i en decideret plan til den næste release. Hver iteration har en længde på mellem 1 og 4 måneder, som ender ud i en del af systemet, der i princippet er klar til at gå i produktion.

2.3.2 Roller

Rollerne i Clear og Orange er naturligvis forskellige i og med Orange fokusere på et større projekt med flere medlemmer. Den største forskel er derfor også, at Clear består af ét team, mens Orange er delt op i mange forskellige teams.

I Crystal Clear er der defineret følgende roller; Sponsor, Senior design-programmør, Design-programmør, Bruger. Senior design-programmøren er nøglepersonen på teamet, mens design-programmørerne kan være et mix mellem erfarende og nyuddannede udviklere[vi].

Udover rollerne er foreslået til Crystal Clear, er der tilføjet følgende roller til Orange; Forretnings ekspert, Bruger ekspert, teknisk facilitator, forretnings analytiker, projektleder, arkitekt, design mentor, ledende design programmør, *UI* designer, forfatter og tester. Et projektmedlem kan dog være tilegnet flere af disse roller, da alle ikke nødvendigvis er fuldtidsstillinger.
Rollerne bliver inddelt i 7 forskellige teams; System planlægning, projekt monitorering, arkitektur, teknologi, funktioner, infrastruktur og ekstern test. Hver af disse grupper skal bestå af en forretnings analytiker, en *UI* designer og 1-3 design programmører [vi].

2.3.3 Praktikker

Følgende regler og praktikker bliver brugt under et trin i udviklingsprocessen for Crystal metoderne [vi]

Staging omfatter planlægning af næste trin i processen. Der skal leveres en fungerende release mellem hver og hver tredje eller fjerde måned. I denne proces vil teamet udvælge, hvilke krav de kan nå at implementere, og dermed planlægge efter det.

Review består af et objektivt gennemgang, af hvad der er blevet udviklet i pågældende iteration.

Tracking vil sikre at fremdriften af projektet bliver målt vha. milepæle og stabilitets faser (meget svingende, svingende eller stabilt). Der bliver altså målt på, om teamet leverer i forhold til milepælsplanen, samt hvilket stadie leverancen er i.

Methodology-tuning technique er en basis komponent i Crystal metoderne. Den gør, at et projekt bruger interviews og workshops, for at tilpasse udviklingsprocessen, og dermed hele tiden evaluerer på processen for at skabe størst muligt output.

Parallelism betyder, at flere teams kan fortsætte og dermed arbejde parallelt. Når stabiliteten ved en leverance bliver vurderet til ”Stabilt”, kan leverancerne for næste opgave startes.

Holistic Diversity Strategy går ud på at splitte store funktionelle teams ud på mindre multi-funktionelle teams. Ideen med denne strategi er at inkludere flere specialister i hvert enkelt team. Dermed opnår man en bedre udnyttelse af den viden, der måtte være i projektet.

Workshops skal afholdes på et regelmæssigt basis. Det kræves, at der holdes pre- og post refleksions workshops, før et trin går i gang.

User viewings skal afholdes to gange i hvert trin. Dette vil bestå af en demonstration af systemet for kunden.

2.3.4 Anvendelse

Der er identificeret følgende restriktioner i anvendelsen af Crystal metoderne[vi]:

Crystal Clear har en forholdsvis stram kommunikations struktur og er derfor kun egnet til, at projektets medlemmer sidder samlet i et kontor. Derudover fungerer metoden kun på mindre teams.
Crystal Orange kræver også at et projekts medlemmer sidder i den samme bygning, for at metoden skal virke. Derudover er der ikke udviklet nogen struktur for de teams, projektet bliver delt op i, hvilket kan blive et stort problem hvis der er 40 projektmedlemmer.

2.4 Kanban

Kanban er et japansk ord, som betyder ”signal kort”. Kanban stammer oprindeligt fra produktions virksomheder, hvor et kort blev brugt til at signalere til en opadgående proces, at der skulle produceres mere. Dog har software udvikling ikke meget med produktion at gøre, men David J. Anderson mener at den Kanban metode han har udviklet kan forbedre produktiviteten, kundetilfredsheden, forudsigeligheden og ikke mindst reducere tidsforbruget[ix].

Helt basalt er et kanban system et antal af kanban kort som til svarer den aftalte kapacitet et system kan klare. Et kort bliver så koblet til et stykke arbejde, og vil fungere som et signal. Et nyt stykke arbejde kan først starte, når der er et ledigt kort. Et ledigt kort kan så blive koblet på et stykke arbejde og følger dette arbejde igennem hele systemet. Når der ikke er flere ledige kort, kan der ikke startes nyt arbejde, og nyt arbejde må derfor vente i kø på, at der bliver et kort ledigt. Dette bliver også kaldt et *pull system*, idet at nyt arbejde bliver trukket (*pulled*) ind i systemet, når det har ledigt kapacitet.

I software udvikling bliver der brugt et virtuelt kanban system til at begrænse det igangværende arbejde. I Kanban implementationen bliver kortene brugt til at repræsentere opgaver. I og med det er et virtuelt system, er det ikke et krav, der bruges fysiske kort. Signalet til at starte nye opgaver, bliver givet ved at trække den virtuelle mængde af igangværende arbejde fra den kapacitet, der er defineret. Dermed finder man ud af, hvornår der er kapacitet til at starte på nye opgaver[ix].

Kanban systemet bliver altså brugt til at begrænse det igangværende arbejde indenfor en fast defineret kapacitet. Dermed kan man balancere det arbejdskrav, der forventes af teamet, og dermed opnå en stabil fremdrift af udviklingen. I og med at nyt arbejde ikke kan påbegyndes før igangværende er helt færdigt, tvinger teamet til at løse problemer, der opstår før nyt arbejde kan påbegyndes. Kanban bliver altså brugt til at optimere processen, og er derfor heller ikke betegnet som en agil udviklingsmetode.

Kanban vil resultere i højere kvalitet i leveranchen samt en bedre præstation blandt udviklerne. I kombination med et bedre flow i udviklingen vil det resultere hurtigere leveringstider og øge forudsigeligheden i projektet[ix].

2.4.1 Proces

Kanban fokuserer på det generelle flow af udviklingen, og opsættes derfor ikke konkrete tidsintervaller for iterationer. Kanban opstiller følgende udfordringer ved de traditionelle iterationer:

- Korte *time-boxe* giver mulighed for at måle fremdrift, men gør at funktionaliteten der udvikles, ikke må have et for stort omfang.
- Kvaliteten af kravene kan mangle kvalitet idet der hele tiden skal planlægges i forhold til næste iteration.
- Kvaliteten af udviklingen afhænger af, om kunden har tid og overskud til at svare på spørgsmål fra udviklerne. Har kunden travlt, kan det resultere i en ringere kvalitet.
- Kvaliteten af test er ofte dårlig, i og med testen ligger til sidst i en iteration og derfor bare skal overstås for at kunne komme i mål med pågældende iteration[ix].

Kanban dropper iterations udvikling og fokuserer på det generelle flow af udviklingen.

Som tidligere nævnt, afskriver Kanban dog ikke en regelmæssig levering til kunden, idet det skaber tillid til, at projektet kan levere resultater. Dog definerer metoden ikke en konkret form for iterationer, for ikke at tvinge udviklingen ned i *timeboxe*, som måske ikke passer. Aktiviteterne bliver dog delt op på prioritering (planlægning og estimering), udvikling og aflevering[ix]. Hver af disse aktiviteter vil have deres egen frekvens og blive tilpasset af teamet. Projektet kan altså have en afleverings frekvens på 4 uger, dvs. de afleverer et stykke færdig kode til kunden hver 4. uge. Prioriterings og udviklingsfrekvensen behøver så nødvendigvis ikke have den samme frekvens, hvis dette ikke giver mening. Det vil fx virke usandsynligt at prioriterings frekvensen skal ske på præcist samme tidspunkt som afleverings frekvensen, og måske er det heller ikke de samme personer, der skal udføre denne[ix]

I bund og grund er der altså ingen faste regler for afleveringsfrekvensen. I nogle tilfælde vil det give mening med en fast aflevering hver måned, og i andre tilfælde vil det give mening med en ad hoc aflevering uden fast defineret interval. Der er fordele og ulemper ved begge dele, men det er helt op til projektet at vurdere disse mod hinanden og vælge den frekvens, der passer bedst til deres projekt.

2.4.2 Roller

Kanban foreskriver ikke nogle konkrete roller for et projekt. Dog vil det være naturligt at have en Product Owner som kan være med til at definere krav.

2.4.3 Praktikker

Kanban præsenterer en række guidelines, som skal følges for at få en succesfuld implementering af Kanban. Disse regler bliver kaldt "*Recipe for Sucess*", og består af følgende[ix]:

Focus on Quality indikerer at teams bruger ofte en stor del af deres tid på at rette fejl. Et eksempel viser at et team har brugt ca. 90% af deres kapacitet på at rette fejl[ix]. Så ved at fokusere på at udvikle et høj kvalitets produkt kan man dermed minimere tiden, der bruges på at rette fejl, og dermed maksimere produktiviteten for projektet. Nogle tiltag indenfor *Focus on Quality* kunne fx være at få professionelle testere til at lave de funktionelle test. Derudover har det vist stor effekt, hvis udviklerne først skriver testen, før de skriver koden til funktionaliteten. Andre tiltag kunne være; kode inspektioner (par programmering, kode gennemgang), analyse og design af løsningen, benytte sig af design patterns, bruge moderne udviklings værktøjer[ix]

Reduce Work-in-Progress er en reducering af det igangværende arbejde (*Work-in-Progress, WIP*), og vil have stor effekt på den endelige kvalitet af systemet. Det viser sig at sammenhængen mellem *WIP* og mængden af opgaver er ikke-lineær. Dvs. at fejl vil forekomme uforholdsmæssigt mange gange og dermed øge mængden af arbejde. Derfor giver det god mening at reducere *WIP* vha. fast definerede rammer for hvor mange opgaver, der må være i gang samtidig.

Deliver Often nævner at en reducering af *WIP* vil også reducere leveringstiden, og en kortere leveringstid vil betyde at der er muligt at levere færdig kode oftere. Dette vil have en positiv effekt overfor fx kunder, som vil få tillid til at projektet nu også leverer det, der er aftalt. Det er derfor vigtigt at planlægge mange små leveringer i stedet for et par store.

Balance Demand against Throughput indebærer, at den hastighed hvormed udviklerne accepterer at tilføje nye krav til pipelinen, svarer til den hastighed hvormed udviklerne kan levere den færdige kode. På denne måde vil vi undgå at skabe en for stor *WIP*, og udviklerne vil føle, de har mere tid, når opgaver ikke hele tiden bliver tilføjet til pipelinen.

Prioritize nævner at hvis de første fire retningslinjer er opfyldt kan det arbejde, der er tilbage blive prioriteret. Dog skal dette kun gøres, hvis teamet er sikre på at funktionaliteten kan blive leveret i den orden, som de bliver prioriteret.

Attack Sources of Variability to Improve Predictability vil hjælpe med at reducere variation, der kan påvirker forudsigeligheden. Effekten af en variation og hvordan man reducerer den inden for en proces, kræver at folk ændrer på måden de arbejder og ændrer adfærd i forhold til processerne. Derfor er dette, det sidste punkt i opskriften på succes, idet det er dette element, der er sværest at implementere.

Udover denne opskrift på succes, har Kanban defineret nogle grundlæggende principper, som supplere opskriften på succes. De består af[ix]:

Visualize Workflow vil synliggør alle stadier som opgaver går igennem. Det kan fx gøres vha. et Kanban board.

Measure and Mange Flow indikerer at det er vigtigt at holde udviklingsflowet i gang hele tiden. Derfor er det vigtigt at holde øje med fremdriften af projektet og dermed sikre, at alt skrider frem som planlagt.

Make Process Policies Explicit vil få hele teamet til at reflektere over effektiviteten. Kanban foreslår at man tænker på en proces som et sæt arbejdsmetoder i stedet for en arbejdsproces. Dermed kan man evaluere på metoderne og se hvad der virker, og ikke virker.

Use Models to Recognize Improvement Opportunities nævner at det er vigtigt at skabe en kultur, hvor forslag til forbedringer drives af alle i teamet, og ikke bare ledelsen.

2.4.4 Anvendelse

Kanban udtrykker ikke nogle specielle begrænsninger for anvendelse af metoden. Metoden kan benyttes til både udviklings, drift- og vedligeholdelses projekter[ix].

2.5 Opsummering

I dette kapitel er den grundæggende teori for de fire agile metoder blevet beskrevet, for at skabe en grundlæggende indsigt i de enkelte agile metoder. I denne forbindelse er der lavet et studie i litteraturen der omhandler en sammenligning af agile metoder, med henblik på at beskrive ligheder. Jeg har derfor undersøgt hvad der tidligere er skrevet om dette emne. Dette er præsenteret i det kommende afsnit.

3 State-of-the-art

Det udførte litteratur studie har været målrettet mod artikler, der har lavet en sammenligning af agile udviklingsmetoder – herunder en analyse af agile metoders egnethed til forskellige projekter. Få artikler har en empirisk undersøgelse af agile metoders egnethed. Dog koncentrer en del artikler sig om en sammenligning af de agile metoder i forhold til den generelle teori.
Den efterfølgende præsentation af tre artikler udgør hvad jeg mener, er state-of-the-art inden for sammenligning af agile udviklingsmetoder. Artiklernes primære fokus ligger alle på en sammenligning af agile metoder, og de to først nævnte artikler adresserer også problemstilling vedrørende hvordan man vælger den korrekte agile metode til sit projekt.

CEFAM: Comprehensive Evaluation Framework for Agile Methodologies (Taromirad et. al., 2009)[x] fremlægger et studie omhandlende behovet for et evaluerings framework til agile metoder, og præsenterer deres bud på et omfattende *framework* til sammenligning af agile metoder, og laver en analyse af XP. Forfatterne har tidligere i [xi] introduceret, hvilke evaluerings kriterier et *framework* bør indeholde, og sammenlignet dette med eksisterende *frameworks*, for dermed at bevise at ingen *frameworks* er fyldestgørende.
I [x] præsenteres en metode, som har til formål at fremhæve ligheder, forskelligheder, *features* og anvendelse for de agile udviklingsmetoder.
Evaluerings kriterierne fra CEFAM bliver delt op i fem grupper; proces, modellerings sprog, agilitet, brugbarhed og cross-context.
Proces evalueringskriterierne er yderligere delt op på seks under kategorier, men fokuserer på procesdelen af en agil metode. Forfatterne mener, at der bør evalueres på modelleringssprog, på trods af at agile metoder ikke tager højde for dette emne. Agilitet evalueringskriterierne er baseret på, hvilke karakteristika der bidrager til en metodes agilitet. Disse kriterier er defineret ud fra det Agile Manifest[i]. Brugbarheds evaluerings kriterierne er baseret på de praktiske aspekter af en agil metode. Dette kriterium adresserer bl.a. projekt specifikke parametre, som kan være værdifulde for projektledere når de skal vælge en metode til et specifikt projekt. Cross-context evaluerings kriterierne fokuserer på den agile metode som helhed, og adresserer de emner der er fundet med mere end en kontekst.
XP er blevet evalueret ud fra de fem kriterier, og resultaterne fremhæver; udviklingsprocessen ikke er entydigt defineret, der bliver ikke taget højde for modellerings sprog, agiliteten er på et acceptabelt niveau og der er ikke mulighed for tilpasning af metoden.
Evalueringsresultaterne er præsenteret ved kvantitative værdier for at skabe værdifulde sammenlignelige resultater. Dog er diskrete værdier også benyttet på steder hvor kvantitative værdier ikke er anvendelige.
Forfatterne konkluderer, at den hierarkiske og kvantitative struktur af CEFAM vil forbedre brugbarheden, og give resultater der er præcise nok til, at en projektleder kan bruge dem til valg af agil metode.

An evaluation of the degree of agility in six agile methods and its applicability for method engineering (Quemer et. al., 2008)[xii] præsenterer et analytisk *framework*, 4-Dimensional Analytical Tool (4-DAT), som er udviklet og brugt til sammenligning af seks agile udviklingsmetoder. Forfatteren mener, at en rapport skrevet ved hjælp af 4-DAT kan hjælpe en organisation med at vælge den korrekte agile metode.
4-DAT evaluerer metoderne ud fra 4 perspektiver; metode *scope*, agilitet, agile værdier og software

proces.
Metode *scope* er defineret ved 8 forskellige *scope* elementer, som er udviklet ud fra nøgle elementer fra eksisterende agile metoders *scope* definition. Agilitet er defineret, ud fra definition af agilitet i [xiii], som fleksibilitet, hastighed, lean, læring og reaktionsevne. Denne dimension bliver brugt til at tjekke agiliteten på både proces- og praktik niveau. Agile værdier er defineret ud fra de agile værdier fra det Agile Manifest[i]. Denne dimension analyserer hvilke praktikker fra de agile metoder, der supporterer de forskellige agile værdier. Software proces er defineret ud fra fire områder, udviklings-, projektledelses-, kontrol- og procesledelses processor.
Seks agile metoder (XP, Scrum, FDD, ASD, DSDM, Crystal) er hver især analyseret ud fra dette *framework*. Quemer's resultater viser, at XP og Scrum er brugbar for små- og medium størrelse projekter, mens Crystal kan bruges på små, medium, og større projekter. Alle metoderne producerer software i hurtige intervaller. Crystal bliver evalueret til at være den mest agile metode på faser, mens Scrum er den mest agile metode på praktikkerne.
Forfatteren afslutter med at konkludere at en evaluering ud fra 4-DAT, både vil give det store billede af agile metoder, samt mere detaljerede værdier og kan dermed hjælpe at træffe en rationel beslutning omkring hvilken agil metode, man skal benytte.

Agile software development: Review and analysis (Abrahamsom et. al., 2002)[viii] præsenterer en sammenligning af ti agile metoder. De omfattede agile metoder i undersøgelsen er; AM, FDD, Crystal, Scrum, PP, ASD, XP, RUP, OSS, DSDM. Formålet med at lave sammenligning er at identificere forskelligheder og ligheder mellem de forskellige agile metoder.
Forfatterne opstiller to perspektiver som de agile metoder bliver analyseret op imod; de vigtigste elementer fra en metode, samt metodens indførelse. Metodens indførelse referer til proces, roller og ansvar, og indførelse og erfaringer for en agil metode.
Resultaterne viser, at agile metoder er designet til brug i små teams af 10 udviklere eller mindre. Metoderne er ikke nødvendigvis brugbare for alle projekter. Skalabilitet er et generelt problem i de forskellige metoder, og vil højst sandsynligt være et problem, når de skal indføres i større organisationer. Forfatterne diskuterer manglen på empiriske studier, der undersøger om de agile metoder er brugbare i forskellige situationer. De understreger, at der er mangel på studier, der undersøger indførelsen og valget af agil metode, som kan være med til at hjælpe organisationer vælge den rigtige metode på det rigtige tidspunkt.
De konkluderer, at de agile metoder deler mange ligheder, men nogle er mere fokuserede end andre. Fx er der forskel på hvilke faser i software udviklingen de forskellige agile metoder understøtter. Derudover er der forskel på, hvor konkrete metoderne er mht. selve udviklings principperne. Her har XP fokus på udviklings principperne, mens Scrum benytter sig af en projektledelses tilgang.

3.1 Opsummering

Artikler hvis hovedformål er at sammenligne agile udviklingsmetoder er her beskrevet. [xii] giver et godt overblik over 6 agile metoder, men giver kun en kort sammenligning af de 6 metoder. [viii] giver en generel sammenligning af 10 agile metoder, og identificerer ligheder mellem disse. Dog er sammenligningen lavet ud fra få kriterier og giver derfor ikke et godt overbliks billede af de agile metoder. [x] laver ikke en decideret sammenligning, men kommer med et forslag til en metode der kan benyttes til sammenligning af

agile metoder. Denne metode vil jeg tage udgangspunkt i til mine undersøgelser, og vil derfor blive gennemgået i detaljer i det kommende afsnit.

4 Metode

Dette afsnit vil gennemgå den undersøgelses metode, der er brugt til analysen af de fire agile metoder, samt benyttet til min undersøgelse af agile metoder i fem private danske virksomheder.

Undersøgelsen af teorien fra de agile metoder vil klarlægge hvilke ligheder der er mellem metoderne. Det forventes at der med den teoretiske undersøgelse, kan skabe resultater der identificerer ligheder, så der kan præsenteres en sammenligning af de agile metoder. Derudover vil en empirisk undersøgelse identificere, hvordan agile metoder er brugt i praksis, og hvilke konkrete projekter en agil metode har virket på. Det forventes at den empiriske undersøgelse, vil skabe resultater, der vil være med til at identificere hvornår en metode har virket på et konkret projektet, og også hvordan en agil metode vælges i virksomhederne.

CEFAM: Comprehensive Evaluation Framework for Agile Methodologies[x]

I dette studie er der taget udgangspunkt I CEFAM metoden til undersøgelsen. En af fordelene der nævnt ved CEFAM er, at ved hjælp af et evaluerings *framework*, at kunne give resultater der er præcise nok til at kunne hjælpe med at udvælge, optage og konstruere agile metoder[x]. Netop problemstillingen med at vælge den rigtige udviklingsmetode til et projekt er et af de centrale elementer i denne rapport.

Projektledere står som regel med udfordringen med at vælge den mest brugbare agile udviklingsmetode til deres projekt. I sådan en situation vil et værktøj, der kan hjælpe med spørgsmålet omkring hvilken agil udviklingsmetode man skal bruge, være uundværlig. I [x] diskuterer forfatterne hvilke krav der er til et evaluerings *framework*, for at kunne analysere agile udviklingsmetoder. De mener en metode skal indeholde; eksisterende udfordringer og specifikke projekt parametre for netop at kunne hjælpe projektledere med at vælge den rigtige agile udviklingsmetode. Forfatterne mener, at en metode til at hjælpe med valget skal identificere; svagheder, kapaciteter, ligheder og forskelle ved de forskellige agile udviklingsmetoder. De mener derfor, at et evaluerings *framework*, der tager højde for disse krav, er den væsentligste ting for at kunne lave en ordentlig analyse af de agile udviklingsmetoder. Derfor er deres mål at opstille et evaluerings *framework* for agile metoder, som netop indeholder de krav.

Selve CEFAM metoden bygger videre på allerede eksisterende *frameworks*, som forfatterne mener, er mangelfulde for at kunne opnå målet med at vælge den rigtige agile udviklingsmetode.

CEFAM opstiller to formål:

1. Fremhæve ligheder, forskelle, features og anvendelse af agile udviklingsmetoder
2. Evalueringen kan hjælpe med at udvælge elementer, og dermed tilpassede en agile udviklingsmetode.

Det første formål til svarer denne rapports formål, nemlig at sammenligne de agile udviklingsmetoder for at kunne identificerer deres ligheder, og finde ud af hvornår de anvendes.

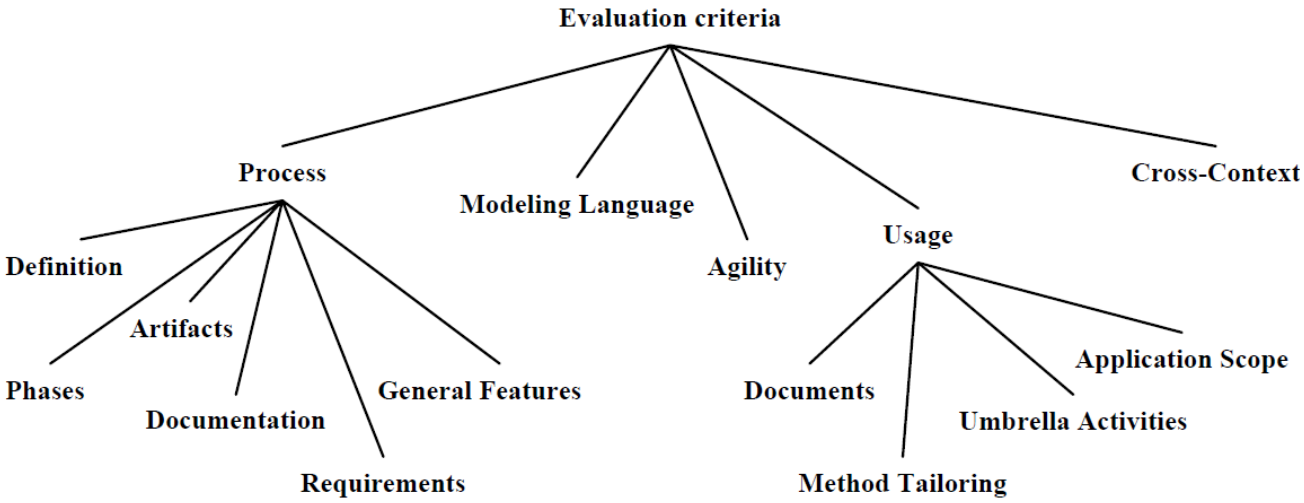
CEFAM opsætter en række evaluerings kriterier, som er dem hele evaluerings *framework'et* er baseret på. Evalueringskriterierne er delt ind i fire overordnede grupper; proces, modelleringssprog, agilitet,

brugbarhed og cross-context. Hver gruppe er yderligere delt op i under grupper, som indeholder evaluerings kriterier som svarer til et specifikt synspunkts i den relevante kontekst.

For at kunne give nogle værdifulde og sammenlignelige evaluerings resultater, er hvert evaluerings kriterium defineret som en kvantitativ måling hvor det giver mening. Dog er der nogle kriterier der ikke egner sig til en kvantitativ måling og, derfor er der i nogle tilfælde brugt diskrete værdier for stadig at kunne sammenligne resultaterne.
Til måling af de kvantitative værdier er der opstillet nogle beskrivende niveauer; uacceptabelt, lavt, medium og høj. I de kvantitative værdier vil resultatet være et decimal tal mellem 0 og 1. De beskrivende niveauer er defineret indenfor følgende intervaller:

$$\text{Uacceptabelt} \leq 0.25$$
$$0.25 < \text{Lavt} \leq 0.5$$
$$0.5 < \text{Medium} \leq 0.75$$
$$0.75 < \text{Høj} \leq 1.0$$

Til de resultater hvor en kvantitativ måling ikke giver mening, vil der blive brugt kvalitative svar.



Figur 5 CEFAM evaluerings kriterier hierarki, taget fra [x]

Evalueringskriteriernes fem hovedområder er yderligere beskrevet herunder.

4.1 Proces

Proces evalueringskriterierne fokuserer på proces delen af den agile udviklingsmetode. Gruppen er yderligere delt op i seks undergrupper:

- *Definition* fokuserer på definitionen af udviklingsprocessen.
- *Faser* fokuserer på udviklingsfaserne og aktiviteterne i disse.
- *Produkter* fokuserer på hvilke produkter der bliver produceret.
- *Krav* analyserer på hvordan krav bliver brugt i udviklingsprocessen.
- *Generelle features* fokuserer på udviklingsprocessen som en helhed.

De agile udviklingsmetoder vil blive analyseret efter proces evalueringskriterierne vha. følgende tabel:

Kriterium	Beskrivelse	Domæne værdier
Definition		
Eksplicithed og entydighed	Er udviklingsprocessen defineret eksplicit og entydigt?	Ja, Nej
Rationale	Er processen blevet rationaliseret ved omfattende og præcise forklaringer?	Ja, Nej (Hvorfor?)
Fuldstændighed	En komplet proces definition indeholder definitioner for: udviklings cyklus, roller, aktiviteter, modellerings sprog, produkter, praktikker/teknikker, regler og paraply aktiviteter.	Forholdet mellem antallet af eksisterende definitioner og en komplet proces definition.
Faser		
Faser i udviklingsprocessen	Hvilke faser er omfattet af udviklingsprocessen? (En komplet definition af faser indeholder: Etablering, Analyse, Design, Implementering, Test, Udrulning, Vedligeholdelse, Support og Post mortem)	Forholdet mellem antallet af omfattede faser og alle beskrevne faser.
Jævn overgang	Er der en jævn overgang mellem faserne?	Ja (teknikker), Nej (eksempler)
Gnidningsfri overgang	Er der nogle gaps mellem faserne?	Ja (teknikker), Nej (eksempler)
Udviklings stil	Hvordan er udviklings stilen?	Iterativ, trinvis, hurtig etc.
Artefakter		
Tilstrækkelige produkter	Producerer udviklingsprocessen de produkter som typisk bliver forbundet med udviklingsaktiviteterne? (Analyse, krav specifikation, design, modellering, dokumentation, test, træning og udrulning)	Forholdet mellem produkt typer der bliver brugt og alle produkter.
Modellering	Inkluderer produkterne modellering?	Ja (Modeller), Nej
Sammenhæng	Supplerer produkterne hinanden?	Høj, Medium (Overlap eksistere, men kan føre til inkonsistens), Lav
Håndgribelighed/Synlighed/Test	Er produkterne håndgribelige, forståelige og kan de testes af slut brugere?	Høj, Medium, Lav
Opfattelse	Hvilken generisk opfattelse får man af produkterne?	Struktureret, Adfærdsmæssig, Funktionel
Abstraktionsniveau	Hvilket abstraktionsniveau er leveret af produkterne?	System/Subsystem/Pakke/intra-object/inter-object, Logisk/Fysisk, Opgave/Proces, Problem/Løsning/Implementation
Standarder	Er der nogen specifikke standarder for produkterne?	Ja (standarder), Nej
Krav		
Krav indsamling	Hvordan bliver krav indsamlet?	Relaterede aktiviteter, roller, produkter
Kravspecifikations format	Hvordan er kravene specificeret?	User story, Feature, Use-case, Usage Scenario etc.
Proces baseret på funktionelle/non-funktionelle krav	Er udviklingsprocessen baseret på kravene?	Ja (teknikker), Nej
Non-funktionelle krav	Hvordan er non-funktionelle krav håndteret?	Teknikker
Sporbarhed	Kan funktionalitet blive sporet til kravene?	Ja (teknikker), Nej
Ændring af krav	Kan man ændre krav i udviklingsprocessen?	Ja (teknikker), Nej
Prioritering af krav	På hvilket grundlag bliver kravene prioriteret?	Arkitekt værdi, Funktionel værdi, Forretnings værdi, udviklings risici.

Generelle features		
Størrelse/Kompleksitet	Størrelse/Kompleksitet er defineret som et overblik over alle elementer i metoden.	Det totale overblik af praktikker, roller, produkter, og faser.
Fuldstændighed	Fuldstændighed er defineret som en funktion af "proces definitionens fuldstændighed", "faser omfattet af udviklingsprocessen" og "tilstrækkelige produkter"	Den gennemsnitlige værdi af; proces definitionens fuldstændighed, faser omfattet af udviklingsprocessen, og de tilstrækkelige produkter.
Praktisk anvendelse	Er processen praktisk anvendelig?	Høj, Medium, Lav
Praktisk mulig	Er udviklingsprocessen praktisk mulig?	Høj, Medium, Lav

Tabel 1 Proces evalueringskriterier

4.2 Modellerings sprog

Forfatterne af metoden mener, at modellerings sprog burde blive brugt i enhver brugbar agil udviklingsmetode. Dog er det de færreste agile udviklingsmetoder, der rent faktisk tager højde for modellerings sprog. I diskussionsafsnittet vil der blive diskuteret baggrunden for, hvorfor forfatterne har valgt at tage dette kriterium med.

Forfatterne har defineret en række kriterier til evaluering af modellerings sprog delen af agile udviklingsmetoder.

De agile udviklingsmetoder vil blive analyseret efter modellerings sprog evalueringskriterierne vha. følgende tabel:

Kriterium	Beskrivelse	Domæne værdi
Sprog	Hvilket modellerings sprog bliver benyttet?	
Nemt at lære og bruge	Er modellerings sproget nemt at lære og bruge?	Ja, Nej
Styrken af sproget	Er modellerings sproget stærkt?	Ja (eksempler), Nej
Håndtering af inkonsistens	Har sproget teknikker for håndtering af inkonsistens?	Ja (teknikker), Nej
Styring af kompleksitet	Har sproget metoder for styring af kompleksitet?	Ja (metoder), Nej

Tabel 2 Modellering sprog evalueringskriterier

4.3 Agilitet

I denne del vil der blive analyseret på hvilke karakteristika der medvirker til en agil udviklingsmetodes agilitet. I forhold til CEFAM metoden er der i dette afsnit lavet en ændring af evaluerings kriterierne. I CEFAM blev en metode evalueret op imod hvilke praktikker, roller og faser der supporterede hhv. fleksibilitet, læring, reaktionsevne og lean. Problemet var her, at der er lagt op til at vurdere dette ud fra den generelle teori af metoderne. Dette vil skabe partiske resultater, da metoderne ikke bliver evalueret op i mod et fælles *framework*, og resultaterne vil derfor ikke kunne bruges. I stedet er der valgt at vurdere de agile metoder op imod værdierne fra det agile manifest[i], for at finde objektive resultater, der kan bruges til en sammenligning.

De agile udviklingsmetoder vil blive analyseret efter de agile evalueringskriterier vha. følgende tabel:

Kriterium	Beskrivelse	Domæne værdi
Hastighed	Hvor hurtigt vil metoden producerer resultater?	1/iterations længe

Bæredygtighed	Er hastighed og kvalitet vedligeholdt indtil slut? Er det kontrolleret og overvåget?	Ja (teknikker), Nej
Individer og samarbejde	Hvilke praktikker supportere individer og samarbejde frem for processer og værktøjer?	Praktikker
Velfungerende software	Hvilke praktikker supportere velfungerende software frem for omfattende dokumentation?	Praktikker
Samarbejde med kunden	Hvilke praktikker supportere samarbejde med kunden frem for kontrakt forhandlinger?	Praktikker
Håndtering af forandringer	Hvilke praktikker supportere at håndtere forandringer frem for at følge en plan?	Praktikker
Lethed og simplicitet	Hvor let og simpel er udviklingsprocessen?	1 / proces kompleksitet
Teknisk kvalitet	Hvordan er teknisk kvalitet målt og kontrolleret igennem udviklingen?	Teknikker og metrikker
Aktiv bruger involvering	Hvor involveret er kunden i udviklingsprocessen?	Roller og ansvarsområder

Tabel 3 Agilitets evaluerings kriterier

4.4 Brugbarhed

I denne del vil der blive analyseret på de praktiske aspekter af en agil udviklingsmetode. Praktiske aspekter ved brugen af en specifik agil udviklingsmetode kunne fx være; Paraply aktiviteter, scope afklaring, skalerbarhed og fleksibilitet. Dette er typiske problemer vedr. brugbarheden af en agil udviklingsmetode, som en projektleder vil støde på.
I dette afsnit stiller CEFAM metoden netop en række generelle parametre op for et projekt. Dette vil muligvis kunne bidrage til besvarelsen af problemstillingen; hvilke slags projekter som den agile udviklingsmetode bedst virker på.

Brugbarhedsevalueringskriterierne er delt op i fire undergruppe:

- *Applikations scope* fokuserer på projekt specifikke parametre, som netop kan bruges af projektlederen når der skal vælges en agil udviklingsmetode til projektet.
- *Paraply aktiviteter* fokuserer på de aktiviteter, der kræves for at få en agil udviklingsmetode til at fungere på et projekt.
- *Method tailoring* fokuserer på aspekter, der bruges til at tilpasse en agil udviklingsmetode.
- *Dokumenter* analyserer på hvilken dokumentation, der er til rådighed for en agil metode.

De agile udviklingsmetoder vil blive analyseret efter brugbarheds evalueringskriterier vha. følgende tabel:

Kriterium	**Sub-kriterium/Beskrivelse**	**Domæne værdier**
Applikations scope		
Projekt	Størrelse	Måneder
	Domæne	Ny udvikling, Data omstilling, Data warehouse, Business Intelligence, Teknologi opdatering, Forskning[xiv]
	Kultur	% trives ved kaos vs. % trives ved orden
	Dynamik	% krav ændring pr. måned
	Kompleksitet	Høj (videnskabelig og kompleks), Medium (forretnings orienteret), Lav (Simpel)
	Kritikalitet (tab pga. fejl)	Komfort, Skønsmæssige midler, Essentielle midler, Liv
	Størrelse	Antal personer

Udviklings team	Uddannelse	-1, 1B, 1A, 2, 3
	Erfaring (i software udvikling)	Høj (8-X år), Medium (5-7 år), Lav (2-4 år)
	Erfaring indenfor domæne	Høj (8-X år), Medium (5-7 år), Lav (2-4 år)
	Erfaring i udviklingssprog	Høj (8-X år), Medium (5-7 år), Lav (2-4 år)
Ergonomi	Fysisk layout	Distribueret, Sammenstillet
Geografi	Antal lokationer for udviklings teamet og kunder	Antal lokationer
Teknisk	Programmerings sprog	Sprog
	Programmerings stil	Simpel, Kompleks
	Abstraktions teknikker	Object-orienteret, Agent-orienteret, etc.
	Obligatoriske udviklings værktøjer	
	Test og debug metoder	
Ledelse	Ledelses team størrelse	Lille, Medium, Stor
	Ledelses erfaring	Høj, Medium, Lav
	Team ledelses tilgang	Centraliseret, distribueret
	Ressource allokerings metode	
	Forretnings kultur	Samarbejdende, ikke-samarbejdende
	Kunde samarbejdsmetode	
	Kvantitative målinger	
Paraply aktiviteter		
Projektledelse	Support til projektledelse i form af: planlægning, kontrollering, overvågning, proces review	Forholdet mellem supporterede aktiviteter mod det samlede antal.
Team ledelse	Har metoden nogle tiltag til processer for team og person ledelse.	Ja (teknikker), Nej
Kvalitets kontrol	Support til kvalitets kontrol teknikker, fx teknisk review, gentagende verifikation og validering,	Ja (teknikker), Nej
Risiko ledelse	Support til risiko ledelses teknikker, fx risiko baseret planlægning, iterativ proces/produkt/plan reviews	Ja (teknikker), Nej
Method tailoring		
Tilpasning og *customizing*	Giver metoden mulighed for at tilpasse metoden i henhold til parametre for projektet.	Ja (metode), Nej
Fleksibilitet	Giver metoden mulighed for at ændre processer og modellerings sprog undervejs?	Ja (hvordan?), Nej
Skalerbarhed	Er metoden brugbar for projekter med forskellige størrelser, kritikalitet, og kompleksitet?	Ja, Nej
Udvidelsesmuligheder	Giver metoden mulighed for nogle udvidelser?	Ja (hvad?), Nej
Integration med andre metoder	Hvis metoden ikke dækker alle aspekter af udviklings cyklussen, bør den bruge en metode til at supplere manglende aspekter	Intet behov, Behov men ikke brugt, Brugt og benyttet,

Dokumenter		
Toturials og trænings dokumenter	Er tutorials og træningsdokumenter tilgængelige?	Ja, Nej
Emperisk bevis	Eksisterer der emperiske beviser?	Ja, Nej

Tabel 4 Brugbarheds evaluerings kriterier

4.5 Cross-context

Evalueringskriterierne i denne gruppe adresserer issues der er til stede i mere end én kontekst. Det kunne fx være, at et issues både går igen i proces og agilitets gruppen, eller at fokus er på den agile udviklingsmetode generelt set.

De agile udviklingsmetoder vil blive analyseret efter cross-context evalueringskriterier vha. følgende tabel:

Kriterium	Beskrivelse	Domæne værdi
Hastighed	Hvor hurtigt vil metoden producerer resultateter?	Iterationslængde
Brugbarhed	Er processen praktisk anvendelig? Er processen praktisk mulig?	Høj, Middel, Lav
Fuldstændighed	Hvor omfattende er metoden?	Den summerede værdi af: ”proces fuldstændighed” og ”projektledelses fuldstændighed”.
Udviklingsproces	Forklarer metoden udviklingsprocessen?	Explicit, Implicit, Nej
Modellerings sprog	Kræver metoden at der bliver brugt modellerings sprog?	Ja, Nej
Begrænsninger	Generelle begrænsninger	Er der nogle generelle begrænsninger for brugen af metoden?

Tabel 5 Cross-context evaluerings kriterier

Denne undersøgelses metode passer godt til dette studies formål, da den netop fokuserer på at sammenligne agile udviklingsmetoder, med henblik på at vælge den rigtige agile metode til et givent projekt. Netop det generelle *framework* der er opstillet vil resultere i, at de agile udviklingsmetoder bliver analyseret ens, og man vil dermed kunne se, hvor de forskellige agile metoder har svagheder og styrker. Resultaterne er opstillet i tabeller for bedre at kunne overskue, hvor de enkelte agile udviklingsmetoder har ligheder. Derudover er der for begrænsningens skyld kun præsenteret de mest interessante elementer fra hvert hovedområde i rapporten. I og med at alle de agile metoder er analyseret op imod det samme *framework*, vil det skabe et godt udgangspunkt for den sammenlignende analyse.

Udover CEFAM analysen er der blevet stillet en række uddybende spørgsmål, for at finde svar på hvorledes projektlederen vælger hvilken agil udviklingsmetode, der skal benyttes til et projekt. Disse spørgsmål kan findes i Appendix A.

Undersøgelsen har taget udgangspunkt i interviews med projektledere. Grunden til dette er at det typisk vil være projektlederen der træffer beslutningen omkring hvilke agile metoder der skal benyttes.

.

4.6 Opsummering

CEFAM metoden fokuserer på at analysere en agil udviklingsmetode ud fra fem kriterier; proces, modellerings sprog, agilitet, brugbarhed og cross-context. Ud fra disse fem kriterier kan der laves en sammenligning af agile metoder, for at kunne identificere ligheder. Undersøgelsen har taget udgangspunkt i denne metode, og resultaterne fra både den teoretiske og empiriske undersøgelse, vil blive gennemgået i efterfølgende afsnit.

5 Resultater

I dette afsnit vil resultaterne fra undersøgelserne bliver beskrevet. Afsnittet er delt op i 2 dele; første del er resultaterne fra den teoretiske undersøgelse af de fire agile metode, mens anden del er resultaterne fra den empiriske undersøgelse ved interviews af fem projektledere fra private danske virksomheder.

5.1 Teori resultater

De agile metoder er blevet analyseret ved CEFAM metoden. Dette vil skabe et overblik over teorien af de forskellige agile metoder, og giver dermed mulighed for at lave en sammenligning af de agile metoder. Resultaterne er opstillet i tabeller, som tager udgangspunkt i 5 definerede hovedområder. Ud fra disse områder forsøges det at klarlægge hvilke sammenhænge der er i de fire agile metoder.

For at skabe et overbliksbillede af de forskellige metoder er der herunder opstillet en tabel, der beskriver hvilke faser, roller, produkter og praktikker der indgår i de forskellige metoder. Herefter vil de agile udviklingsmetoder blive analyseret efter CEFAM's fem hovedområder med henblik på at sammenligne de agile metoder.

Metode	**XP**	**Scrum**	**Crystal**[1]	**Kanban**
Faser	1. Exploration 2. Planning 3. Iterations to release 4. Productionizing 5. Maintenance 6. Death	1. Pre-game 2. Development 3. Post-game	1. Planning 2. Development 3. User use-able release	-
Roller	7. Tester 8. Interaction Designer 9. Architect 10. Project Manager 11. Product Manager 12. Executive 13. Technical Writer 14. User 15. Programmer	4. Scrum Master 5. Product Owner 6. Scrum Team 7. Customer 8. Management	4. Sponsor 5. Senior design-progammer 6. Design-programmer 7. User	-
Produkter	16. Story cards 17. Task list 18. Customer acceptance test 19. Unit test	9. Product Backlog 10. Sprint Backlog 11. Release burndown 12. Sprint backlog burndown	8. Release sequence 9. Common object model 10. User manual 11. Test cases 12. Migration code	1. Visualize Workflow
Praktikker	20. Sit Together 21. Whole Team 22. Informative Workspace 23. Energized work 24. Pair Programming 25. Stories 26. Weekly Cycle	13. Sprint planning 14. Sprint 15. Release planning 16. Sprint review 17. Sprint retrospective 18. Daily Scrum	13. Staging 14. Review 15. Tracking 16. Methodology-tuning 17. Parallelism 18. Holistic Diversity Strategy	2. Focus on Quality 3. Reduce WIP 4. Deliver Often 5. Balance demand against Throughput 6. Prioritize 7. Attack Sources of Variability to

[1] Da Crystal består af en familie af metoder, vil der her for begrænsningens skyld, kun blive fokuseret på praktikkerne fra Crystal Clear.

	27. Quarterly Cycle 28. Slack 29. Ten-Minute Build 30. Continuous Integration 31. Test-First Programming 32. Incremental Design		19. Workshops, 20. User viewings	Improve Predictability, 8. Measure and Manage Flow 9. Make Process Policies Explicit 10. Use Models to Recognize Improvement Opportunities

Tabel 6 Overblik over faser, roller, produkter og praktikker

I de følgende afsnit er de væsentligste resultater fra hvert af de fem hovedområder blevet samlet.

5.1.1 Proces

Ud fra resultaterne i Appendix B, er der opstillet følgende tabel som opsummerer de væsentligste elementer fra proces evalueringen af de agile udviklingsmetoder.

	XP	Scrum	Crystal	Kanban
Definition	0,875	0,875	0,875	0,5
Faser	0,66	0,55	0,55	0,22
Artefakter	0,625	0,5	0,75	0,125
Krav	Kunden	*Product Owner*	Forretningsekspert	-
Generelle features	Faser: 6 Roller: 9 Produkter: 4 Praktikker 13 I alt: 32	Faser: 3 Roller: 5 Produkter: 4 Praktikker: 6 I alt: 18	Faser: 3 Roller: 4 Produkter: 5 Praktikker: 8 I alt: 20	Faser: 0 Roller: 0 Produkter: 1 Praktikker: 9 I alt: 10

Tabel 7 Proces

Definition er et udtryk for, hvor komplet en agil udviklingsmetode er defineret. Den kvantitative værdi er udregnet ud fra, hvilke definitioner en agil metode har, og sat i forhold til en komplet definition. En komplet definition af en proces, ifølge [x], bør indeholde definitioner for; udviklingscyklus, roller, aktiviteter, modellerings sprog, produkter, praktikker, regler og paraply aktiviteter. XP , Scrum og Crystal har definitioner for præcis de samme elementer nemlig; udviklingscyklus, roller, aktiviteter, produkter, praktikker, regler og paraply aktiviteter og opnår derfor den samme værdi[ii][iv][vi]. Kanban har den mindst definerede proces med definitioner for; aktiviteter, produkter, praktikker og paraply aktiviteter[ix]. Definitionen siger dog ikke noget om indholdet af definitionen, men giver blot en værdi af hvor mange af elementer, der er defineret. Så selvom resultaterne ligger meget ens, er det nødvendigvis ikke ensbetydende med, at indholdet også er ens. Den kvantitative værdi vil skabe arbitrære resultater, og vil derfor ikke give brugbare resultater til en sammenligning. Dette vil blive diskuteret yderligere i diskussions afsnittet.

Faser er et udtryk for, hvor mange faser der er omfattet af udviklingsprocessen. Her opstiller CEFAM metoden 9 generiske faser, som de agile metoders faser bliver evalueret op i mod. Den kvantitative værdi er antallet af omfattede faser sat i forhold til alle definerede faser. De 9 generiske faser er defineret som; Etablering, Analyse, Design, Implementering, Test, Udrulning, Vedligeholdelse, Support og Post mortem[x]. Scrums tre definerede faser(*pre-game, development, post-game*), vil omfatte; analyse, design, implementering, udrulning og død. Crystal Clear vil også have 5 faser omfattet i udviklingsprocessen nemlig; analyse, design, implementering, udrulning og død. XP's udviklingsproces vil omfatte analyse, design, implementering, test, vedligeholdelse og død, og opnår derfor den mest komplette udviklingsproces.

Kanban foreskriver ikke nogle konkrete faser som fx Scrum, men processen vil omfatte implementering af koden samt udrulning[ix]. Igen vil den kvantitative værdi give vilkårlige resultater, og dette vil derfor bliver diskuteret i afsnit 6 .

Artefakter er et udtryk for de produkter, der bliver produceret i forbindelse med; analyse, kravspecifikation, design, modellering, dokumentation, test, træning og udrulning. Den kvantitative værdi er udregnet ud fra om artefakter er omfattet af de enkelte elementer fx analyse, og sat i forhold til alle elementerne. Her er det kun Crystal der kommer op på et forholdsvis højt niveau på 0,75. Crystal producerer artefakter forbundet med; analyse, kravspecifikation, design, dokumentation, test og træning[vi]. XP producerer artefakter i forbindelse med analyse, kravspecifikation, design, dokumentation og test[ii]. Kanban producerer kun produkter i forbindelse kravspecifikationen og opnår derfor en lav værdi[ix]. Scrum producerer artefakter i forbindelse med analyse, kravspecifikation, design og dokumentation[iv]. Den kvantitative værdi vil endnu engang kunne give arbitrære resultater ved udelukkende at lave en sammenligne ud fra værdien, derfor er dette kriterium yderligere diskuteret i efterfølgende afsnit.

Krav er en summering af, hvilke aktiviteter og roller de agile metoder har i forbindelse med indsamling af krav, definering og prioritering af krav. I XP vil kunden stå for at skrive *user stories*, som så vil blive revideret i fællesskab med udviklerne i starten af hver iteration. I Scrum er det *Product Owner,* som har det overordnede ansvar for *backlog'en*, og vil også står for prioriteringen. Scrum teamet og kunden vil dog også indgå i dialog omkring de enkelte *user stories*.
I Crystal er det ikke helt så klart defineret hvordan krav håndteres. Det er op til projektet at definere hvem der skal være forretningseksperten, som eventuelt kunne være ansvarlig for kravene. Dog kræves der i Crystal, at der oprettes et krav dokument til håndtering af kravene. Kanban har ikke nogle definitioner for hvem, og hvordan krav skal håndteres.

Generelle features er en opsummering af hvilke faser, roller, produkter og praktikker hver agil metode indeholder. En komplet oversigt over disse kan findes i Tabel 6.

5.1.2 Modellerings sprog

Ingen af de 4 agile metoder forholder sig til brugen af modellerings sprog. Tabellen for analysen af modellerings sprog er derfor blot tom.

	XP	Scrum	Crystal	Kanban
Modelleringssprog	-	-	-	-
Nemt at lære og bruge	-	-	-	-
Styrken af sproget	-	-	-	-
Håndtering af inkonsistens	-	-	-	-
Styring af kompleksitet	-	-	-	-

Tabel 8 Modellerings sprog

5.1. 3 Agilitet

Ud fra resultaterne i Appendix B, er der opstillet nedenstående tabelsom opsummerer de væsentligste elementer fra agilitet evalueringen af de agile udviklingsmetoder.

	XP	Scrum	Crystal	Kanban
Hastighed	Iterationer på 1 uge	Iterationer på 1 uge - 1 måned	Iterationer på 1 - 3 mdr.	-
Individer og samarbejde	1. *Sit Together* 2. *Whole Team* 3. *Pair Programming*	1. *Scrum teams* 2. *Sprint planning* 3. *Daily Scrum*	1. *Holistic Diversity Strategy* 2. *Parallelism* 3. *User viewings*	-
Velfungerende software	1. *Test-First proramming* 2. *Continuous integration* 3. *Ten-Minute Build* 4. *Pair Programming* 5. *Weekly Cycle*	1. *Sprint* 2. *Sprint Review*	1. *Review* 2. *Tracking* 3. *User viewings*	1. *Focus on Quality* 2. *Reduce WIP*
Samarbejde med kunden	1. *Sit Together* 2. *Whole Team* 3. *Stories* 4. *Weekly Cycle*	1. *Product Backlog* 2. *Sprint Backlog* 3. *Sprint planning*	1. *Staging* 2. *Review* 3. *User viewings*	1. *Prioritize* 2. *Deliver Often*
Håndtering af forandringer	1. *Weekly Cycle* 2. *Slack*	1. *Sprint review* 2. *Sprint retrospective* 3. *Daily Scrum*	1. *Workshops* 2. *Methodology-tuning technique*	1. *Measure and Manage Flow* 2. *Visualize Workflow*
Lethed og simplicitet	1/32	1/18	1/20	1/10

Tabel 9 Agilitet

Hastighed beskriver hvor hurtigt en agil udviklingsmetode vi levere resultater. Her er der fokuseret på længden af iterationer, da metoderne vil levere færdig kode i slutningen af hver iteration, der i princippet er klar til at gå i produktion. Ingen af metoderne giver en helt konkret længde på en iteration. Dog opstiller alle, undtagen Kanban, et forslag til interval som iterationerne bør være indenfor, da metoden i stedet fokuserer på det generelle flow af udviklingen. XP og Scrum ligger tæt på hinanden i iterations længder, dog foreslår Scrum, at iterationer kan vare helt op til en måned, hvor XP foreslår ugentlige iterationer. Crystal kalder iterationer for trin, men ideen er det samme som en iteration. Crystal foreslår trin på 1 til 3 måneders varighed. Kanban foreskriver ikke noget konkret interval som en iteration skal holde sig indenfor. Dog anbefales det, at der planlægges en regelmæssig cyklus for leverancer for at skabe tillid hos kunden[ix].

Individer og samarbejde (frem for processer og værktøjer) indikerer hvilke praktikker der supporterer samarbejde. I XP anbefales det, at teamet sidder i samme lokale, både for at gøre kommunikationen lettere, men også for at man kan opnår følelsen af, at man er "et team". Derudover er parprogrammering en væsentlig del af samarbejdet i XP, hvor to udviklere sidder ved samme computer og skriver koden. I Scrum er der også fokus på selve Scrumteamet. Dette er også baseret på samarbejde, i og med at medlemmerne kan have forskellige kompetencer, men tilsammen vil have de færdigheder, der skal til for at løse et problem. Til *Sprintplanning* mødet vil kunden og *Srum Masteren* i fællesskab planlægge næste sprint. *Daily Scrum* mødet vil også fremme samarbejdet i Scrum teamet, da hver udvikler præsenterer hvad han har lavet, og hvilke udfordringer han står over for. I Crystal vil *Holistic Diversity Strategy* splitte store funktionelle teams ud på mindre multi-funktionelle teams, for dermed at opnå et bedre samarbejde og

udnyttelse af viden. *Parallelism* vil fremme, at flere teams kan arbejde i parallelle spor. *User viewings* vil fremme samarbejdet med kunden ved at lave to demonstrationer af systemet i løbet af en iteration. Kanban har ingen praktikker, der supporterer samarbejdet.

Velfungerende software(frem for omfattende dokumentation) indikerer, hvilke praktikker der fokuserer på at udvikle velfungerende software. XP er den metode med flest praktikker, der understøtter dette punkt. *Test-First programming*, *Continuous integration*, *Ten-Minute Build*, *Pair programming* og *Weekly Cycle* vil alle sammen bidrage til at udvikle velfungerende software frem for at fokuserer på dokumentation af systemet. Scrum har to praktikker der understøtter dette punkt, Sprint og *Sprint Review*. I Sprintet vil al software blive udviklet, men der er ikke fokus på, hvordan dette skal gøres. *Sprint Reviewet* vil gennemgå de opnåede resultater i Sprintet med kunden, for at sikre at det opnåede resultat stemmer overens med kundens forventninger. Crystal vil ligesom Scrum have et *Review* i slutningen af hver iteration, som vil gennemgå resultaterne. Derudover vil det være to *user views* i løbet af iterationen, hvor systemet vil blive demonstreret for kunden. Begge disse praktikker er med til at sikre, at det udviklede system stemmer overens med kundens forventninger. *Tracking* vil være med til at sikre milepælsplanen bliver fulgt, og at der bliver leveret de ønskede resultater. Kanban benytter sig af to praktikker, der vil hjælpe med at udvikle velfungerende software; *Focus On Quality* og *Reduce WIP*. Fokus på kvaliteten vil minimere den tid, der skal bruges på at rette fejl senere hen, og dermed sikrer et højere kvalitets produkt. *Reduce WIP* vil sikrer, at kun at vist antal opgaver er under udvikling hele tiden, og dermed kan der fokuseres bedre på igangværende arbejde og opnå en højere kvalitet.

Samarbejde med kunden (frem for kontraktforhandling) beskriver hvilke praktikker, der vil hjælpe til et bedre samarbejde med kunden. I XP vil *Sit Together* sikre, at hele teamet sidder samlet inklusiv kunden. Det vil sikre en bedre kommunikation og bedre mulighed for samarbejde. Derudover vil *Whole team* forsøge at skabe følelsen af at være ét team, kunden vil altså være en integreret del af teamet. *Stories* vil sikre at funktionalitet er skrevet, så kunden også kan forstå dem, og dermed være med til at udvælge hvilke der skal implementeres i staten af *Weekly cycle*. I Scrum vil *Product*- og *Sprint backlog* være med til at fremme samarbejdet med kunden ved at lade kunden prioritere disse lister. Derudover vil kunden deltage i *Sprint planning* for at være med til at diskutere hvilke stories der er vigtigst at implementere i næste Sprint.
I Crystal vil *Staging* fremme samarbejdet med kunden ved at planlægge den næste iteration i fællesskab med kunden. *Review* og *User viewings* vil præsentere, hvad der er blevet udviklet for kunden, og dermed sikre at det er de rigtige krav, der bliver udviklet. I Kanban vil *Deliver Often* sikrer samarbejdet med kunden ved at levere færdig kode så ofte som muligt. Derudover vil *Prioritize* være med til at prioritere opgaverne for at kunne skabe størst værdi for kunden.

Håndtering af forandringer (frem for fastholdelse af en plan) beskriver hvilke praktikker der vil være med til håndtere forandringer i projektet. I XP vil *Weekly Cycle* være med til at planlægge en uge frem I tiden, og vil derfor medvirke til at kunne omstille sig til forandringer. *Slack* vil inkludere nogle få opgaver i planen, som kan droppes undervejs hvis man kommer I tidsnød. I Scrum vil *Sprint planning* være med til at håndtere forandringer i projektet. Der vil kun blive planlagt i fx to uger frem i tiden, hvilket vil gøre det nemmere at ændrer planlægning ved uforudsete udfordringer. *Sprint Review* vil være med til at identificere, hvad der er udviklet, og kunden har her mulighed for evt. at foreslå ændringer eller forbedringer af systemet. *Sprint retrospective* vil reflektere over selve processen, og kan være med til at tilpasse processen. *Daily Scrum* vil hjælpe med at identificerer de konkrete udfordringer udviklere står over for i deres arbejde, som fx kan

skyldes forandringer. I Crystal vil *Methodology-tuning technique* hele tiden være med til at evaluere på processen, og dermed være med til at håndtere forandringer. Workshops vil også være med til at reflektere over processen både før og efter en iteration. I Kanban vil *Measure and Manage Flow* indikere, hvis der er for mange opgaver i gang ad gangen, og giver dermed mulighed for at tilpasse sig forandringerne. *Visualize Workflow* vil give et visuelt billede ad dette.

Lethed og simplicitet er blot en summering af antallet af faser, roller, produkter og praktikker der indgår i de agile metoder. Antallet siger dog ikke meget om den agile metode, da mange af elementerne blot er vejledende. Derfor giver det ikke god mulighed for sammenligning.

5.1.4 Brugbarhed

Ud fra resultaterne i Appendix B, er der opstillet følgende tabel som opsummerer de væsentligste elementer fra brugbarhedsevalueringen af de agile udviklingsmetoder.

	XP	**Scrum**	**Crystal**	**Kanban**
Scope	Mindre teams, alle størrelse projekter	Mindre teams, alle størrelse projekter	Kan bruges på alle størrelser	-
Projektledelse	0,75	1,0	1,0	0,75
Kvalitets kontrol	*Pair programming, Continuous integration, Sit Together, Test-First programming*	*Sprint Review*	*User viewings, Review*	*(Focus On Quality)*
Tilpasning	Ja	Ja	Ja	Ja
Fleksibel	Ja	Ja	Ja	Ja
Skalerbarhed	Ja	Ja	Ja	Ja
Udvidelse	Ja	Ja	Ja	Ja
Integration	-	Muligt	Muligt	Muligt

Tabel 10 Brugbarhed

Scope er en sammenfatning af *Applikations scope* fra resultaterne i Appendix B. XP definerer at metoden er egnet til projekter med mindre teams. En vigtig faktor er også, at udviklerne sidder sammen, så det er hurtigt og nemt at kommunikere og koordinere. Scrum foreslår også, at teamsne ikke skal have en størrelse på meget over 10 personer.
Crystal kan bruges på alle størrelser af projekter. Dog består Crystal af et sæt af metoder, og afhængigt af størrelsen vil man benytte en forskellig metode fra Crystal familien.
Kanban nævner ikke nogle begrænsninger for brugen af metoden, eller ideelle teamstørrelser.

Projektledelse resultaterne er defineret som en kvantitativ værdi. Denne er fundet ud fra om der er nogle praktikker, der bidrager til planlægning, kontrollering, overvågning, og proces review. Dog kan værdierne ikke sammenlignes, da ikke alle metoderne har fx overvågning i metoden. Derfor vil værdierne være arbitrære og ikke kunne bruges til en sammenligning. På den baggrund er der opsat nedenstående tabel for at lighederne i projektledelsen kan blive diskuteret yderligere i diskussions afsnittet.

	XP	**Scrum**	**Crystal**	**Kanban**
Planlægning	1. *Weekly Cycle* 2. *Quaterly Cycle*	1. *Sprint planning*	1. *Staging*	1. *Reduce WIP*
Kontrollering	3.*Weekly Cycle*	2. *Sprint review*	2.*Review*	2. *Measure and*

		3. *Burndown charts* 4. *Daily Scrum*	3. *User viewings*	*Manage Flow*
Overvågning		5. *Sprint Backlog* 6. *Product Backlog*	4. *Tracking*	3. *Visualize workflow*
Proces review	4. *Quaterly Cycle*	7. *Sprint retrospective*	5. *Methodology-tuning*	

Tabel 11 Projektledelse

Kvalitets kontrol beskriver, hvilke praktikker i metoderne der understøtter kvalitetskontrollen. XP har flest praktikker, der fokuserer på kvaliteten af systemet. Parprogrammering vil sikrer en god dialog ved programmeringen og kan dermed være med til at løse komplekse problemer på den mest optimale måde. *Test-First Programming* og *Continuous integration* fokuserer begge på at teste systemet grundigt og ofte inden der ændres i koden, hvilket også vil have stor indflydelse på kvaliteten. *Sit Together* vil også være med til at skabe god kvalitet, i og med problemer hurtigt og effektivt kan løses ved dialog med kunden eller andre udviklere. I Scrum vil der være et *Sprint review* efter hvert Sprint, hvor det opnåede resultat vil blive vurderet. På samme måde vil der i Crystal være et *review* i slutningen af hver iteration, men også flere *user viewings* i løbet af iterationen. I Kanban bliver der blot nævnt, at det er vigtigt at fokusere på kvaliteten, men ikke konkret hvilke tiltag der skal benyttes.

Tilpasning indikerer om den agile metode giver mulighed for at tilpasse metoden i forhold parametre i projektet. Scrum kan tilpasses alt efter team størrelse, dvs. hvis teamet er for stort kan dette opdeles i flere separate teams som alle kører Scrum. Crystal definerer tydeligt, at det kan være en god idé at supplere metoden med elementer fra XP eller Scrum, men at det er op til projektet at vælge, hvad der passer ind i deres kontekts. Kanban afskriver heller ikke muligheden for at benytte elementer fra andre metoder, og foreslår at det kan være en god idé at holde daglige stand-up møder ligesom *Daily Scrum*. XP giver også mulighed for tilpasning i form af, at hvis et problem er for stort, kan det evt. splittes op og løses af flere forskellige teams i stedet.

Fleksibel er en betegnelse for, om den agile metode giver mulighed for at ændrer processer undervejs. I Crystal, Scrum og Kanban giver i høj grad mulighed for at ændrer diverse processer undervejs. I Scrum kan iterationslængden fx tilpasses i løbet af projektet, mens man i Kanban kan ændre på værdien af WIP, hvis der fx kommer en ny udvikler til teamet. Crystal ligger også optil at kunne ændrer processer undervejs, fx i form at tilpasse elementer til metoden undervejs. XP giver også mulighed for tilpasning undervejs i processen, fx kan et eller flere teams tilføjes undervejs afhængigt at projektets størrelse og opgavernes kompleksitet.

Skalerbar og *Udvidelse* hænger sammen, og er et udtryk for om metoden er brugbar for projekter med forskellige størrelser, kritikalitet og kompleksitet, samt om det er muligt at tilføje udvidelser til metoden. XP giver klart udtryk for, at metoden vil fungerer godt på mindre teams. Dog vil XP også fungere på større projekter, der bliver delt op i flere forskellige mindre teams. Scrum, Crystal og Kanban kan også skaleres til større projekter. Scrum kan skaleres til store projekter, hvor et stort team vil deles op i mange små teams med under 10 udviklere på hvert team, præcis ligesom XP. Crystal familien definerer som sagt flere forskellige metoder ud fra antallet af personer på projektet og kritikaliteten, så denne metode kan også skaleres. Kanban siger ikke noget konkret om størrelsen af projekter denne metode virker på, men i og med metoden fokuserer på at optimere udviklings processen ved at mindste det igangværende arbejde, vil metoden kunne benyttes på de fleste projekter.

Integration indikerer om den agile metode giver mulighed for at bruge elementer fra en anden agil metode til at supplere eventuelle manglende aspekter. XP nævner ikke brugen af, eller behovet for at integrere med andre agile metoder. Scrum, Crystal og Kanban nævner alle, at der kan tilføjes elementer fra andre metoder, hvis det passer ind i projektets kontekst.

5.1.5 Cross-context

Ud fra resultaterne i Appendix B, er der opstillet følgende tabel som opsummerer de væsentligste elementer fra cross-context evalueringen af de agile udviklingsmetoder.

	XP	Scrum	Crystal	Kanban
Hastighed	1 uge	1 uge – 1 mdr.	1 mdr. – 3 mdr.	-
Brugbarhed	-	-	-	-
Fuldstændighed	0,8125	0,9375	0,9375	0,625
Udviklingsproces	Implicit	Implicit	Eksplicit	Implicit
Begrænsninger	Teamstørrelse, Virksomhedskultur	Teamstørrelse	Projektet skal være placeret i samme bygning for at metoden virker.	-

Tabel 12 Cross-context

Hastighed er blot en opsummering af, hvor hurtigt en metode vil producere færdig kode. XP vil være den hurtigste metode og anbefaler, at der afleveres færdig kode hver uge. Scrum anbefaler et iteraitons interval på mellem 1 uge – 1 mdr. Crystal foreslår en del længere iterationer på op til 3 mdr. Kanban anbefaler at aflevere færdig kode ofte for at sikre et godt forhold til kunden. Dog fokuserer Kanban på det generelle flow i udviklingen, og benytter sig derfor ikke af iterationer.

Brugbarhed er ikke vurderet her, da det vil være en subjektiv vurdering. Derimod er brugbarheden vurderet i næste afsnit ud fra de projekter der er blevet undersøgt.

Fuldstændighed er en definition af den generelle proces fuldstændighed og projektledelses fuldstændighed. Men i og med tallene fra proces fuldstændigheden og projektledelses fuldstændigheden er arbitrære, kan resultaterne ikke sammenlignes. Dette vil blive diskuteret yderligere i diskussions afsnittet.

Udviklingsproces er resultaterne fra om metoderne har en eksplicit eller implicit måde at forklare selve udviklingsprocessen på. Både XP, Scrum og Kanban har en implicit måde at forklare udviklingsprocessen på.

Begrænsninger er de generelle begrænsninger, som der er opsat for nogle af metoderne. XP har den begrænsning, at metoden fungerer bedst ved mindre teams, og derudover nævner Beck, at hvis virksomhedskulturen strider imod XP's principper vil det heller ikke være muligt at implementere XP. Crystal Clear har en begrænsning på teamstørrelse på max. 6 personer, og kræver også at teamet sidder i samme rum. Derudover har Crystal Orange den begrænsning, at de forskellige teams skal sidde i samme bygning for at metoden vil fungere. Scrum nævner også, at teamstørrelsen heller ikke skal overstige ca. 9 personer. Kanban nævner ikke nogle specifikke begrænsninger for metoden.

5.2 Empiri resultater

Med udgangspunkt i CEFAM frameworket er der udarbejdet et interview, til undersøgelse i fem private danske virksomheder. Dette har til formål at finde ud af hvordan en agil metode er benyttet i praksis, hvilke

projekter en agil metode har virket på, samt hvordan beslutningsgrundlaget for valg af metode er. Hovedfokus i den empiriske undersøgelse er dog at undersøge om man ud fra projekters parametre kan forudsige, hvornår en bestemt metode vil virke. På den baggrund vil der i dette afsnit udelukkende blive fokuseret på brugbarheds evalueringskriteriet. Alle projekterne er dog analyseret efter CEFAM metoden, og resultaterne kan findes i Appendix D. Derudover kan alle resultaterne fra interviewet findes i Appendix E.

Min undersøgelse er baseret på interviews med 5 projektledere i forskellige private danske virksomheder. Projekterne er blot præsenteres som P1, P2, P3, P4 og P5. En kort oversigt over projekterne kan findes i Appendix C. Den agile metode brugt i de forskellige projekter har hovedsageligt været Scrum.

Ud fra resultaterne i Appendix E, er der opstillet følgende tabel som opsummerer de væsentligste elementer fra brugbarhed evalueringen af de fem projekter. Først vil kriterierne kort blive præsenteret ud fra Tabel 13, hvorefter de vil blive analyseret grundigere hver deres underafsnit.

	P1	**P2**	**P3**	**P4**	**P5**
Scope	Se Tabel 14	Se Tabel 14	Se Tabel 14	Se Tabel 14	Se Tabel 14
Projektledelse	1,0	1,0	1,0	1,0	1,0
Kvalitets kontrol	Kode *reviews*	QA funktion	Løbende test fra forretningen	Eksternt kode *review*	*Inspections*
Tilpasning	Ja	Ja	Ja	Ja	Ja
Fleksibel	Ja	-	Ja	-	Ja
Skalerbarhed	Ja	Ja	Ja	-	Ja
Udvidelse	Ja	Ja	Ja	Ja	Ja
Integration	Brugt og benyttet	Brugt og benyttet	Brugt og benyttet	Brugt og benyttet	Brugt og benyttet

Tabel 13 Brugbarhed

Scope er vurderet mere detaljeret end teori afsnittet, for at kunne vurdere hvilke projekter en agile metode er blevet benyttet på. Der er derfor opsat en række parametre for et projekt ud fra [x], og illustreret nedenfor i Tabel 14, hvor *scope* vil blive gennemgået i detaljer.

Projektledelse er vurderet ud fra hvilke praktikker der bidrager til planlægning, kontrollering, overvågning og proces review. Dette vil blive gennemgået yderligere i afsnit 5.2.2.

Kvalitetskontrol er en vurdering af hvilke praktikker projekterne benyttede sig af for at sikre kvaliteten i udviklingen.

Tilpasning vil illustrere i hvor høj grad projekterne har tilpasset den agile metode til at passe deres kontekst.

Fleksibel er en betegnelse for, om projekterne har ændret på nogle af processerne i løbet af projektet.

Skalerbar og *udvidelse* er to sider af samme sag, og er et udtryk for om projekterne har benyttet sig af udvidelser eller skaleret metoden til fx flere teams.

Integration indikerer om projekterne har benyttet andre praktikker fra andre agile metoder.

5.2.1 Scope

	P1	P2	P3	P4	P5
Domæne	Ny udvikling	Ny udvikling	Data integration	Ny udvikling	Produktudvikling
Længde	9 mdr.	17 mdr.	18 mdr.	7 mdr.	15 mdr.
Kultur	60%	10%	30%	50%	70%
Dynamik	40%	10%	30%	10%	20%
Kritikalitet	Skønsmæssige /Essentielle midler	Essentielle midler	Skønsmæssige/ Essentielle midler	Essentielle midler	Liv
Størrelse (antal personer)	10	80	7	15	28

Tabel 14 Scope

Domæne er en vurdering af indenfor hvilket område projektet kan karakteriseres. P1, P2 og P4 ligger alle sammen indenfor samme domæne som er betegnet som "Ny udvikling". Ifølge [xiv] er dette også de typiske projekter, som agile udviklingsmetoder bliver benyttet på. *Længde* er, hvor lang tid projektet var fra den første fase til projektet officielt blev afsluttet.
Kulturen er vurderet ud fra, hvor godt projektet trives med kaos, dvs. hvis det vurderes at projektet trives 0 procent med kaos, vil alt være forudsigeligt og intet vil komme som en overraskelse. Resultatet er ud fra en subjektiv vurdering fra projektlederen. Ud fra [xv] vurderes det, at jo større procentdel projektet trives med kaos, jo bedre vil projektet være egnet til at arbejde efter en agil udviklingsmetode. Dog ses det på resultaterne at kulturen er meget forskellig for de fem projekter. P5 har den højeste værdi på 70%, hvilket vil sige, at det et meget uforudsigeligt projekt. Dette skyldes i høj grad kompleksiteten, da projektet udvikler et innovativt produkt til forsvarsindustrien. Kravene var som sagt fast defineret for projektet, men kompleksiteten i disse krav viste sig til tider at være store, og dermed skabte det en meget uforudsigelig kultur for projektet. I skarp kontrast er P2, som har en forholdsvis forudsigelig kultur. Dette skyldes blandt andet at det er en videreudvikling af et allerede kendt system, og kompleksiteten derfor ikke er så høj.
Dynamik er vurderet ud fra hvor mange procent kravændringer der bliver foretaget hver måned. Resultaterne er dog her en subjektiv vurdering fra hver af de adspurgte projektledere, så resultatet er altså hverken præcist eller upartisk. Dog nævner P2, P4 og P5, at de havde en fast defineret kravspecifikation inden projektet gik i gang, og havde derfor ikke mange ændringer til kravspecifikationen undervejs. P1 og P3 havde ikke et helt så klart *scope* for projektet som de tre andre. Her blev flere krav lavet om, tilføjet og slettet undervejs, hvilket resulterede i en del flere kravændringer pr. måned
Kritikalitet er en vurderingen af, hvad det kan risikere at koste projektet, hvis det færdige projekt indeholder fejl. Her skal det nævnes, at P5 udvikler et produkt til forsvarsindustrien, og derfor har en kritikalitet der i værste tilfælde kan koste liv. P2 udviklede et nyt system til bankverdenen, og en kritisk fejl ville derfor i værste tilfælde betyde tab af helt essentielle midler. P4 udviklede et nyt webbaseret system, der skulle distribuere midler til private, derfor ville en kritisk fejl her også betyde et tab af essentielle midler. P1 og P3 var begge udvikling af et internt system til virksomheden og derfor er kritikaliteten i disse projekter ikke ligeså høj.
Størrelse er antallet af personer der er med på projektet. Det ses tydeligt, at Scrum kan bruges på flere forskellige størrelser af projekter. P2 var det største af projekterne med 80 deltagere, og brugte derfor muligheden for at skalere Scrum ved at indføre mange Scrum-teams, som arbejdede på samme projekt. Dette benyttede P4 og P5 sig også af, mens P1 og P3 blot bestod af et enkelt Scrum-team

5.2.2 Projektledelse

Alle projekterne har en forholdsvis høj værdi i *projektledelse,* hvilket vil sige at de benytter sig helt eller delvist af elementer, der bidrager til planlægning, kontrollering, overvågning og proces review. Igen kan den kvantitative værdi ikke benyttes til en sammenligning, så derfor er nedenstående tabel sat op der illustrerer hvilke praktikker fra projekterne der bidrager til projektledelsen.

	P1	P2	P3	P4	P5
Planlægning	1. *Sprint planning* 2. *Pre-sprint planning*	1. *Sprint planning*	1. *Sprint planning*	1. *Sprint planning*	1. *Sprint planning*
Kontrollering	2. *Burndown charts*	2. *Burndown charts*	2. *Burndown charts*	2. *Burndown charts* 3. *Sprint review*	2. *Burndown charts* 3. *Sprint review*
Overvågning	3. *Sprint Backlog* 4. *Product Backlog*	3. *Sprint Backlog* 4. *Product Backlog*	3. *Sprint Backlog* 4. *Product Backlog*	3. *Sprint Backlog* 4. *Product Backlog*	3. *Sprint Backlog* 4. *Product Backlog*
Proces review	5. *Sprint retrospective*	5. *Sprint retrospective*	5. *Sprint retrospective*	5. *Sprint retrospective*	5. *Sprint retrospective*

Tabel 15 Projektledelse i de 5 projekter

5.2.3 Kvalitets kontrol

P1 benytter sig af kode *reviews* i forbindelse med *Kvalitets kontrol.* Her bliver der afsat tid til at gennemgå koden til en *user story* med en anden udvikler, hvilket kan karakteriseres som en form for par programmering. P2 havde en QA funktion, der stod for alt kvalitetskontrollen af projektet. P3 benyttede sig af løbende test for kunden, for både at sikre kvaliteten men også at der blev leveret de krav, som kunden ønskede. P4 havde et eksternt bureau til at lave *review* på al koden inden projektet blev afsluttet. P5 benyttede sig af såkaldte *inspections*, som kan sammenlignes med kode *reviews*, men hvor fokus også ligger på at forklare tankerne bag koden, så man lærer af processen.

5.2.4 Tilpasning

Tilpasning vurderer om metoden kan tilpasses ud fra specifikke projekt parametre. Alle projekterne har lavet ændringer og tilføjelser til metoden og dermed tilpasset Scrum efter deres kontekst. Herunder er lavet en sammenligning af Scrum praktikkerne benyttet i de fem undersøgte projekter mod det totale antal af praktikker i Scrum[xvi]. Her ses det tydeligt, at alle projekterne har tilpasset deres Scrum metode i mere eller mindre grad, for at den skal passe projektet.

Projekt	P1	P2	P3	P4	P5
Agil metode anvendt	Scrum	Scrum	Scrum	Scrum	Scrum
Sammenlignet med	Scrum	Scrum	Scrum	Scrum	Scrum
Totale praktikker i metoden	18	18	18	18	18
Anvendte praktikker i	10	10	14	15	13

projektet*					
% Agil metode brugt	55%	55%	78%	83%	72%

Tabel 16 Tilpasning

* Her er udelukkende talt Scrum praktikker med. Projektets samlede antal praktikker kan findes i Tabel 17, men dette tal vil indikere alle praktikker brugt i projektet, og dermed være et højere tal da alle projekterne benyttede sig af praktikker fra andre metoder. Resultaterne her viser altså hvor meget de har tilpasset metoden, de nævner de bruger, til at passe projektets kontekst.

Ingen af projekterne hævdede dog at bruge Scrum 100% efter teorien så derfor er resultaterne ikke så overraskende. Det er dog værd at nævne, at P4 er det projekt med mindst erfaring med Scrum, men de havde en Scrum coach tilknyttet, som kan være derfor at de opnår den højeste procentdel af Scrum benyttet.

5.2.5 Fleksibel, skalerbarhed, udvidelse og integration

Fleksibel vurderer om det er muligt at ændrer processer undervejs, og hvad der eventuelt er blevet ændret i sådanne tilfælde. P1 valgte at tilføje kode *reviews* undervejs i processen, for at sikre bedre kvalitet i koden. I P3 havde man oprindeligt 1 måneds sprint, men valgte at skifte til 2 ugers Sprint, da det viste sig at planlægningen blev for besværlig, når man skulle planlægge 1 måned ud i fremtiden. P5 havde oprindeligt 3 *Scrum Masters* - 1 til hvert Scrum team. Dog valgte man at skifte til én *Scrum Master,* som virkede som *Scrum Master* i alle tre Scrum teams.

P2, P3 og P5 har benyttet *skalerbarheden* i Scrum ved at tilføje flere Scrum teams på samme projekt.

Udvidelse indikerer om den agile metode giver mulighed for at udvide denne. Da alle projekterne har benyttet Scrum, og har udvidet metoden med elementer der passer til deres kontekst er der svaret ja ved alle projekterne.

Alle projekterne har benyttet sig af *integration* med andre metoder, også selvom de måske ikke var klar over det.

5.2.6 Beslutningsgrundlag

I Appendix A kan der findes de supplerende spørgsmål, som blev stillet til testpersonerne. Her blev spurgt omkring beslutningsgrundlaget for valget af en agil metode.

I P1 blev valg af metode besluttet ud fra personlige erfaringer fra projektledelsen.

P2 blev besluttet fra projektets side. Her var der nogle personer, der havde god erfaring med Scrum, og derfor blev det foreslået som udviklingsmetode til styregruppen af projektet.

I P3 blev valget af metode også besluttet ud fra personlige erfaringer med Scrum blandt teammedlemmerne.

P4 arbejdede sammen med et eksternt bureau som udelukkende arbejdede med Scrum, og derfor også krævede at samarbejdspartnere benyttede sig af denne metode. Derfor blev der er ansat en Scrum coach til at lære projektet metoden, og følge op på processerne undervejs.

P5 valgte også metoden ud fra personlige erfaringer fra projektet. I virksomheden er der politik for, at det er helt op til projekterne selv at vælge den metode, de finder mest brugbar.

6 Diskussion

Resultaterne vil indledningsvist blive diskuteret med henblik på at besvare de hypoteser som er blevet fremlagt i problemformuleringen for dette studie, og dernæst vil CEFAM metoden diskuteres.

6.1 Sammenligning af de agile metoder

I forrige afsnit er der opstillet resultater fra de fire agile metoder. Men hvad er egentlig lighederne i disse fire metoder? Denne vurdering er lavet ud fra teorien af de fire agile metoder.

Tabel 6 viser et overblik over de fire agile metoders faser, roller, produkter og praktikker. Denne tabel er ikke en del af CEFAM frameworket, men den er en vigtig del for at kunne få et godt overblik over elementerne de fire agile metoder.

I følgende underafsnit vil fokus være på at diskutere resultaterne med henblik på at lave en sammenligning af de agile metode. For at skabe et overblik over metodernes ligheder, præsenteres de fem overordnede kategorier fra CEFAM frameworket enkeltvis:

6.1. 1 Proces

Den kvantitative værdi for *definition* giver ikke belæg for en sammenligning, da tallet blot giver udtryk for, hvor mange elementer, der er defineret. Værdien kan derfor heller ikke bruges til en sammenligning, i det de samme kvantitative værdier ikke nødvendigvis vil betyde, at metoderne har de samme definitioner. Resultaterne vil derfor her være arbitrære og ikke mulige at sammenligne.

Faser kriteriet giver heller ikke noget brugbart resultat, der kan bruges til en sammenligning. I og med det at to værdier på fx 0,5 ikke vil sige noget om, at det er de samme faser, der indgår i metoderne. Derfor kan værdierne ikke uden videre sammenlignes, og der må derfor findes en anden metode til sammenligning af faserne. I stedet for en kvantitativ værdi kunne kriteriet være splittet op i de 9 forskellige faser for at se, hvordan de forskellige faser kommer til syne i hver af metoderne. En kvalitativ værdi ville her give et bedre grundlag for en sammenligning. Fx ville man kunne identificere, hvordan implementeringsfasen indgår i de forskellige metoder, og sammenligne dette.

Artefakter kriteriet strider i første omgang mod, hvad det agile manifest mener omkring omfattende dokumentation, som ikke giver værdi for selve udviklingen. I dette kriterium vil en højere værdi være bedre, hvilket ikke stemmer overens med fx XP's tilgang til artefakter, som går ud på at fjerne alle artefakter der ikke bidrager til selve udviklingen. Ud fra CEFAM metoden vil en høj værdi i artefakter kriteriet altså være mere positivt end en lav, hvilket overhovedet ikke giver mening set i forhold til de forskellige agile metoders tilgang til artefakter.
Til dette kriterium ville en kvalitativ vurdering af hvordan de enkelte artefakter indgår i metoden igen kunne give værdi for en sammenligning. Ved fx at se på hvordan en kravspecifikationen indgår i de forskellige metoder, ville man identificere, og sammenligne de forskellige tilgange til dette.

Krav kriteriet viser, at der er en del sammenhænge i håndteringen af krav i to af metoderne. Scrum og XP har stort set samme proces for udarbejdelse af *user stories* og prioritering af disse. Begge metoder lader kunden skrive *stories*, og inden hver iteration starter, vil der i samarbejde med kunden blive udvalgt, hvilke

stories der skal indgå i den kommende iteration. Crystal og Kanban nævner ikke noget om håndteringen af krav, Crystal nævner blot at et kravdokument er obligatorisk.

En analyse af proces kriteriet fra CEFAM metoden giver altså ikke det store overblik over de forskellige agile metoders proces. De mange kvantitative værdier der benyttes siger ikke meget konkret om processen, og giver ikke mulighed for en korrekt sammenligning. I [xii] er sammenligningen af processen baseret på udviklingsproces, projektledelses proces, support proces og procesledelse. Her bliver hver af de agile metoders praktikker analyseret op imod disse fire elementer, og der bliver identificeret konkrete praktikker fra de agile metoder der bidrager til disse processer. Denne fremgangsmåde skaber bedre resultater der giver mulighed for en sammenligning af processen.

6.1.2 Modellerings sprog

En væsentlig fællesnævner blandt alle fire agile metoder er, at ingen af metoderne foreskriver behovet for at bruge modellerings sprog. Det er altså ikke eksplicit defineret i nogle af metoderne at brugen af modellering er obligatorisk.

Der er ingen af de agile udviklingsmetoder, der rent faktisk tager højde for modellerings sprog. I [xi] analyserer de tidligere *frameworks* til sammenligning af agile udviklingsmetoder, og analyserer på hvilke mangler disse har. Ud fra deres studier i tidligere *frameworks* opstiller de en række nye kriterier for, hvilke elementer der bør indgå i et *framework* til analyse af agile udviklingsmetoder. Her har de netop fokus på, at modellerings sprog er en væsentlig part af en metode som bør indgå i analysen, også selvom de agile metoder ikke direkte siger noget omkring modellerings sprog.

Taromirad og Ramsin mener stadig, at det er en væsentligt punkt, på trods af ingen agile metoder foreskriver dette. Til en analyse af teorien bag de agile metoder giver denne kategori ikke nogen værdi. Dog kan dette punkt være interessant i en empirisk undersøgelse, for at se om modellerings sprog bliver tilføjet de agile metoder.

6.1.3 Agilitet

Hastighed kriteriet viser at XP, Scrum og Crystal alle fokuserer på udvikling i iterationer. Dog er der forskel på længden af iteraitoner. XP foreslår ugentlige iterationer. Scrum og Crystal definerer begge et interval de foreslår iterationerne ligger inden for. Scrum foreslår at iterationer vare 1 uge – 1 måned, mens Crystal foreslår 1 måned – 3 måneder. Kanban går ikke ind for denne type af iterations udvikling idet metoden fokuserer på det generelle flow i stedet. Kanban foreslår, at der fokuseres på selve flowet af udviklingen, ved at gøre selve udviklingsprocessen synlig vha. et Kanban board, og begrænse det igangværende arbejde.

Den oprindelige fremgangsmåde til at vurdere agilitet, foreslået af CEFAM metoden, viste sig at give arbitrære værdier. Derfor er der i dette afsnit benyttet en anden fremgangsmåde end hvad CEFAM metoden benytter sig af. Her er de agile metoder blive vurderet op imod de fire agile værdier, for at kunne skabe en sammenligning af hvilke praktikker i de fire agile metoder der supportere de forskellige værdier. På denne måde er der opnået et bedre grundlag for en sammenligning.

Praktikkerne, der understøtter *Individer og samarbejde* har ikke nogle direkte fællesnævnere.
Dog findes der sammenhænge i praktikkerne, der understøtter *Velfungerende software*. *Weekly cycle* fra XP er den uge, hvor udviklingen vil finde sted, i lighed med Sprint fra Scrum som blot vil have en anden

længde. Derudover vil *Weekly cycle* også indeholde et *Review,* som det også ses at der benyttes i Scrum og Crystal.

Der er også nogle ligheder i praktikkerne, der supporterer *Samarbejde med kunden. Weekly cycle* vil få kunden til at vælge, hvilke *stories* der skal udvikles i kommenende iteration, på samme måde som man vil gøre under *Sprint planning* i Scrum. Crystal bruger lidt samme metode i *Staging*, men her er det blot op til hele teamet at planlægge og ikke nødvendigvis kun kunden.

Håndtering af forandringer viser, at der bl.a. er sammenhæng mellem *Sprint retrospective* og *Methodology-tuning technique* som begge vil reflektere over processen og evt. tilpasse sig forandringer.

6.1.4 Brugbarhed

Scopet for både XP og Scrum er at de er bedst egnede til mindre teams. Dog er det også den sammenhæng at begge metoder kan skaleres til at fungerer på et stort projekt med mange teams, på samme måde som Crystal.

Det grundlæggende problem med de kvantitative værdier i CEFAM metoden går igen i *Projektledelses* resultaterne, som igen er arbitrære. XP og Kanban opnår her den samme kvantitative værdi, da de begge dækker 3 ud af 4 opsatte parametre for projektledelse. Men i og med det ikke er de samme parametre, der går igen i metoderne, kan resultatet ikke sammenlignes, og den kvantitative værdi vil altså være forkert. Dog giver Tabel 11 her mulighed for at lave en sammenligning, da den netop illustrerer hvilke praktikker fra metoderne, der supportere de enkelte dele af projektledelsen. På den måde kan praktikkerne diskuteres og sammenlignes, i det de bliver vurderet op i mod et enkelt kriterium ad gangen. Dette har skabt nogle bedre kvalitative resultater, i stedet for den kvantitative værdi der ikke kunne sammenlignes. Tabel 11 viser bl.a., at der er sammenhænge i planlægningen i XP, Scrum og Crystal. *Weekly cycle, Sprint planning* og *Staging* indeholder alle et møde lige før en iteration, der har til formål at planlægge den næste iteration. Derudover er der også sammenhænge i kontrolleringen, hvor XP, Scrum og Crystal alle benytter sig af et *review* til at følge op på, hvad der er udviklet. Der er også en fællesnævner i, at i XP, Scrum og Crystal benytter alle sig af et proces review vha. hhv. *Quaterly Cycle*, *Sprint retrospective* og *Methodology-tuning*. Det er forskel på, hvor ofte disse *reviews* bliver lavet.

Det ses, at XP er den metode, der fokuserer klar mest på kvalitetskontrollen, hvilket også stemmer godt over ens, med at det er den metode, der har flest praktikker, der understøtter *"Velfungerende software frem for omfattende dokumentation".*

Alle metoderne giver mulighed for at tilpasse metoden undervejs, dog er det kun Scrum, Crystal og Kanban som nævner, at der med fordel kan benyttes elementer fra andre metoder.

6.1.5 Cross-context

Mange af resultaterne I cross-context Tabel 12, er blot en opsummering eller sammenfatning af flere resultater fra tidligere, og vil ikke blive diskuteret igen.

Fuldstændigheden er den summerede værdi fra proces fuldstændigheden og projektledelsens fuldstændighed. Men som tidligere nævnt, er begge disse resultater arbitrære, hvorfor dette resultat heller ikke vil kunne benyttes til en sammenligning.

En fælles begrænsning for XP og Scrum er, at teamstørrelserne anbefales at være af mindre størrelser.

I mine resultater har jeg udvalgt elementer fra CEFAM *frameworket* for at kunne skabe overblik. [x] mener at dette *framework* er komplet, og det netop er nødvendigt med et så omfattende *framework* for at lave en sammenligning. Dog er der en gennemgående tildens til at de kvantitative værdier benyttet i metoden skaber vilkårlige resultater som ikke er brugbare til en sammenligning. Derfor er der kun opnået gode sammenlignelige resultater steder hvor der er brugt et kvalitativt udgangspunkt.

En brugbar sammenligning for en virksomhed skulle gerne skabe et overblik over de enkelte agile metoder, og fremhæve de væsentligste ligheder mellem metoderne. En analyse med CEFAM *frameworket* giver en god mulighed for at sammenligne de agile metoder i detaljer, dog giver de kvantitative værdier ikke mulighed for en sammenligning. Stort set alle kvantitative værdier foreslået af CEFAM metoden er ofte uden decideret indhold og vil skabe arbitrære resultater. Fx vil udregning af den kvantitative værdi for projektledelse, blot være baseret om der er nogle praktikker der supportere planlægning, kontrollering, overvågning og proces *review*. Værdien vil altså intet sige om hvordan metoderne bidrager til de forskellige elementer. Derudover vil værdien heller ikke kunne sammenlignes. Et eksempel kunne være at en agil metode har praktikker for praktikker der supportere planlægning og kontrollering, mens end anden agil metode har praktikker der supportere overvågning og proces *review*. Begge de agile metoder vil så opnå den samme værdi, uden at de har nogle praktikker til fælles, og den kvantitative værdi vil derfor ikke kunne sammenlignes.

6.2 Hvornår virker hvilke metoder på hvilke projekter?

De agile metoder giver få guidelines til hvornår en metode vil virke. Men ét er hvad teorien siger vil virke, noget andet er, hvad fungerer i virkeligheden. Derfor er der lavet en undersøgelse med henblik på at finde ud af, hvilke projekter en konkret agil metode har virket på. Oprindeligt var ideen at finde flere forskellige projekter, som hver benyttede sig af en forskellig agil metode. Det viste sig dog hurtigt, at Scrum er den klart mest udbredte metode, og derfor var det kun muligt at finde projekter i virksomheder, som hovedsageligt benyttede sig af Scrum metoden. Stort set alle projekterne benyttede sig dog af elementer fra andre metoder, og P5 benyttede sig i en kort periode udelukkende af Kanban.

Der er i Tabel 14 vist de fem undersøgte projekters parametre, for at kunne analysere om der er en tendens for hvornår en agil metode har virket på et konkret projekt. Domæne parameteren viser at tre af projekterne består af udvikling af et nyt system, mens ét fokuserer på dataintegration og det sidste på produktudvikling. Denne parameter siger dog ikke meget konkret om et projekt, og kan derfor tolkes på mange måder. Dynamik parameteren viser i dette tilfælde, at Scrum er blevet benyttet både på projekter med et forholdsvist fast *scope* samt et mere dynamisk *scope*. Dog er resultatet en subjektiv vurderingen fra projektlederens side, hvorfor resultatet ikke vil være præcist. Kritikalitet parameteren viser ingen stor sammenhæng, men det interessante her, at P5 har et kritikalitets niveau på tab af liv. Ifølge [xv] vil et projekt med sådan et kritikalitets niveau være bedre egnet til en traditionel vandfaldsmetode med mere stram styring end en agil udviklingsmetode. Kulturen parameteren igen en subjektiv vurdering fra projektlederen, og viser heller ingen sammenhæng mellem projekterne. Der er heller ingen sammenhæng i projekternes størrelse eller længde.

Ud fra en sammenligning af de fem projekters parametre kan der ikke ses nogen signifikant sammenhæng mellem parametrene. Parametrene kunne ønskes at være mere objektive i stedet for en subjektiv

vurdering fra projektlederen i flere tilfælde. Derudover vil projektets parametre ikke nødvendigvis være statiske igennem hele projektets levetid. Der kan ske ændringer i teamstørrelse, projektet kan blive forlænget eller uforudsete udfordringer kan resultere i et mere dynamisk miljø. Derfor er tilpasning og udvidelse af metoden i høj grad ligeså vigtig som at kunne vælge den rigtige metode fra start.

Det ses også at alle projekterne benyttede sig i høj grad tilpasning og udvidelse af deres agile metode. P5 benyttede sig i en kort periode udelukkende af Kanban metoden. I en kort periode hvor udviklingen udelukkende bestod af at rette fejl, blev Kanban indført. Grunden til dette var, at fejlene blev rettet, som de blev meldt ind, og det gav derfor ikke mening af køre sprints med 2 ugers planlægning ad gangen, da man ikke vidste, hvad der var af opgaver 2 uger i fremtiden. Dette er et perfekt eksempel på at tilpasse sin agile metode til de udfordringer projektet står overfor. Ved at have et godt overblik over de enkelte agile metoder og ved, hvor de kan bidrage, kan man tilpasse eller skifte ens metode til projektet og de udfordringer man møder undervejs.

Men spørgsmålet er så, om et projekt med samme parametre som fx P5 også vil have succes med at benytte sig af Scrum og evt. Kanban. Det kan med rimlighed antages, at et projekt med de samme parametre som P5, i en anden virksomhed, ikke vil opleve de samme omstændigheder, udfordringer eller eksterne faktorer. Der kan derfor ikke konkluderes på at det lige præcis er projektets parametre der alene er afgørende for hvornår en metode vil virke.

6.3 Hvordan vælger man den rigtige metode til sit projekt?

Når der ikke kan svares konkret på, hvornår en metode virker på et konkret projekt, hvordan kan man så vælge den rigtige metode. I de undersøgte projekter er en generel tendens til, at der ikke er nogle retningslinjer eller guidelines til, hvordan man vælger den rigtige metode til sit projekt. Dvs., der er retningslinjer for, at det er op til projektet selv at beslutte, hvilken metode de vil benytte. Dette er selvfølgelig en fornuftig idé, da alle projekter ikke nødvendigvis vil have gavn af den samme metode. Dog er beslutningsprocessen i projekterne stort set udelukkende baseret på erfaringer med den pågældende udviklingsmetode fra tidligere projekter. Der er selvfølgelig ikke noget negativt i at basere beslutningsprocessen på erfaringer, men dog at det er en smule mangelfuldt, når erfaringerne udelukkende omfatter én enkelt metode.
Det virker ikke, som om der er lagt de helt store tanker i valget af metode med blot af ”Det virkede på mit gamle projekt, og derfor virker det vel også på det nye”. Her savner en mere kritisk tilgang til valget af metode.

En af grundende til det lige netop er Scrum projekterne har valgt, skyldes at det er den metode som flest har gode erfaringer med. Et af projekterne udtalte at de ikke viste der fandtes andet end Scrum – de troede agil udvikling udelukkende var Scrum. Scrum er måske det bedste valg i mange tilfælde pga. simple processer, den høje grad af fleksibilitet og den tilpasning, som metoden tilbyder. I disse tilfælde mangler dog en mere kritisk tilgangsvinkel til valget af agil metode. Men for at dette er muligt, kræver det man er bekendt med, at der findes andre agile metoder og ved hvad de kan tilbyde. Her kommer en sammenligning af de agile metoder til sin ret.

Resultaterne fra interviews viser at der er en mangel på overblik blandt interviewdeltagerne over, hvilke forskellige agile metoder, der findes. Derfor er beslutningsprocessen udelukkende baseret på erfaringer med en enkelt agil metode, dog med undtagelse af P5. Dette projekt formår at tilpasse og vælge en ny

metode i løbet af projektet baseret på skiftende omstændigheder. Der er altså ikke nødvendigvis én rigtig agil metode gennem hele projektet, og ud fra resultaterne diskuteret i 0, kan der sagtens tilføjes og tilpasses praktikker fra andre agile metoder.

CEFAM metoden som jeg har benyttet til min undersøgelse, hævder også at levere sammenlignende resultater af agile metoder, der er præcise nok til at kunne hjælpe med at vælge den rigtige metode til et projekt. Dog er der plads til væsentlige forbedringer af metoden. De kvantitative værdier der er foreslået giver ikke mulighed for sammenligne resultaterne, hvilket er et stort problem. Der er dog ingen tvivl om, at et overblik og sammenligning af de agile metoder vil bidrage til at kunne vælge den rigtige agile metode.

Teorien opsætter meget få guideline for, hvornår metoden kan benyttes, og der er mangel på empiriske undersøgelser på præcis dette emne[x]. Derfor bør fremtidige undersøgelser fokusere på at undersøge mange forskellige projekter, der har benyttet de forskellige agile metoder.

6.4 Fremtidige studier

Et videre studie af sammenligning af agile metoder, kunne med fordel beskæftige sig med at sammenligne metoderne ud fra et mere kvalitativt grundlag end CEFAM metoden foreslår. Derudover kan agile metoders egnethed til forskellige projekter, undersøges i et større omfang. I første omgang bør det undersøges om der kan opstilles nogle objektive projekt parametre, som kan bruges til en undersøgelse af agile metoders brug i virksomheder. Derudover bør det også undersøges hvilke andre faktorer der spiller ind på valget af metode, så der ikke alene fokuseres på projektets parametre. Datagrundlaget for undersøgelsen bør dog være forholdsvist stort for muligvis at kunne sige noget signifikant om brugen af en konkret metode til et specifikt projekt.

7 Konklusion

I denne rapport er der blevet udarbejdet en sammenligning af fire agile udviklingsmetoder ud fra CEFAM metoden.
XP, Scrum, Crystal og Kanban er blevet sammenlignet ud fra fem dimensioner; proces, modellerings sprog, agilitet, brugbarhed og cross-context.
Flere kriterier fra CEFAM metoden viste sig at give arbitrære værdier, hvorfor ikke alle kriterier fra metoden kunne bruges til en sammenligning. Det ses dog at der er flere ligheder omkring hvordan krav bliver håndteret i flere af de agile metoder. Derudover er der ingen af metoderne eksplicit definerer guidelines vedr. modellerings sprog.
Ud fra analysen af agilitet ses det at der i flere af metoderne anbefaler at der udvikles i iterationer. Det ses også at flere af metoderne har ligheder i de praktikker der understøtter de fire overordnede agile værdier.
På bagrund af analysen af brugbarheden er det vist at der er ligheder mellem hvornår de agile metoder anbefaler at kunne bruges . Derudover er det vist, at alle de fire metoder giver mulighed for skallering og tilpasning af metoden. Til sidst i sammenligningen er det vist, at begrænsningerne for de fire agile metoder er få og ikke velkendte.

Der er blevet undersøgt hvornår en konkret agil metode virker på et projekt, ud fra en undersøgelse af projekter i fem private virksomheder. Her ses det at alle fem projekter benyttede sig af Scrum, men at metoden er blevet tilpasset med praktikker fra andre agile metoder i alle fem tilfælde. Der har ikke kunne vises en signifikant sammenhæng mellem de fem projekternes parametre. Derfor kan der ikke konkluderes på at den agile metode, de har benyttet, vil virke på et specifikt projekt. Det ses at en agil metodes egnethed til et konkret projekt, i høj grad også afhænger af tilpasningen, som projektet vil tilføje den agile metode. Der er vist at der ikke udelukkende ud fra projekters entydige parametre kan bestemmes hvornår en agil metode vil virke på et konkret projekt. Det ses, at beslutningsprocessen i virksomhederne er baseret på enkelt personers personlige erfaring med en bestemt agil metode. Derudover konkluderes det også at en sammenligning af agile metoder vil kunne være med til at danne et overblik over mulighederne og dermed optimere beslutningsprocessen.

Til slut konkluderes det at de foreslåede kvantitative værdier fra CEFAM metoden, ikke vil kunne bidrage til en sammenligning mellem agile metoder, idet de skaber arbitrære resultater.

Referencer

[i]. K. Beck et. al., *Manifesto for Agile Software Development,* http://agilemanifesto.org/, 2001

[ii]. K. Beck, *Extreme Programming Explained: Embrace Change, Second Edition,* Addison-Wesley, 2004

[iii]. K. Beck, *Extreme Programming Explained: Embrace Change,* Addison-Wesley, 2000

[iv]. K. Schwaber and J. Sutherland, *The Scrum Guide – the official rulebook,* Scrum guide, 2010

[v]. K. Schwaber et al., *Agile Software Development with SCRUM*, Prentice Hall, 2002

[vi]. A. Cockburn, *Agile Software Development: The Cooperative Game (2nd Edition),* Pearson Education, 2007

[vii]. A. Cockburn, *Agile Software Development,* Addison-Wesley, 2001

[viii]. P. Abrahamson et. al., *Agile software development methods: Review and analysis*, VTT Publication, 2002

[ix]. David J. Anderson, *Kanban: Succesful Evolutionary Change for Your Technology Business,* Blue Hole press, 2010

[x]. M. Taromirad et. al., *CEFAM: Comprehensive Evaluation Framework for Agile Methodologies,* IEEE, 2009

[xi]. M. Taromirad et. al., *An Appraisal of Existing Evaluation Frameworks for Agile Methodologies*, IEEE, 2008

[xii]. A. Quemer et. al., *An evaluation of the degree og agility in six agile methods and its applicability for methods engineering,* Information and Software Technology 50 280-295, 2008

[xiii]. A. Quemer et. al., *Measuring agility and adoptability of agile methods: A 4-Dimensional Analytical Tool,* IADIS, 2006

[xiv]. Agile Project Types, *http://www.ambysoft.com/surveys/agileProjectTypes2011.html,* Ambysoft, 2011

[xv]. B. Boehm, R. Turner, *Balancing Agility and Discipline:A Guide for the Perplexed*, Addison Wesley, 2003.

[xvi] D. Strode, *The Target Environment for Agile Methods,* Australasian Conference on IS, 2008

Appendix A

Uddybende spørgsmål til interview

Til spørgsmålet omkring hvordan metoden bliver valgt:

Hvilke forventninger havde i til implementeringen af metoden?

Hvordan blev disse forventninger opfyldt?

Hvordan blev det besluttet af det netop var den metode I skulle bruge?

Hvad var den afgørende faktor for at det netop er den metode i bruger?

Har I erfaring med andre metoder?

Kunne det forestilles at kombinere elementer fra flere metoder?

Appendix B

Resultater fra analyse

I dette appendix findes resultaterne fra analysen af de fire agile metoder ved CEFAM metoden.

	XP	**Scrum**	**Crystal**	**Kanban**
Proces				
Definition				
Eksplicithed og entydighed	Nej (Processen er beskrevet ved nogle eksempler)	Nej (Det er op til det enkelte projekt at definerer hvad der skal indgå i udviklingsprocessen)	Ja (Udviklingsprocessen er udført ved at følge en række beskrevede guidelines for; politiske standarder, work products, local matters, tools, standards og roller).	Nej (Metoden vil hjælpe til med at optimere en eksisterende proces, og vil altså ikke have klare guidelines for en konkret proces)
Rationale	Nej (ingen detaljeret proces beskrivelse, for at kunne tilpasses)	Nej (For at kunne tilpasse sig en fleksibel verden, er der ikke nogle præcise forklaringer)	Ja	Nej
Fuldstændighed	0,875 (7/8) XP har definitioner for: - Udviklingscyklus - Roller - Aktiviteter - Produkter - Praktikker - Regler - Paraply aktiviteter	0,875 (7/8) Scrum har definitioner for: - Udviklingscyklus - Roller - Aktiviteter - Produkter - Praktikker - Regler - Paraply aktiviteter	0,875 (7/8) Crystal har definitioner for: - Udviklingscyklus - Roller - Aktiviteter - Produkter - Praktikker - Regler - Paraply aktiviteter	0,5 (4/8) Kanban har definitioner for: - Aktiviteter - Produkter - Praktikker - Paraply aktiviteter
Faser				
Faser i udviklingsprocessen	0,66 (6/9) Faser omfattet: - Analyse - Implementering - Test - Udrulning - Vedligeholdelse - Post mortem	0,55 (5/9) Faser omfatter: - Analyse - Design - Implementering - Udrulning - Post mortem	0,55 (5/9) Faser omfattet: - Analyse - Design - Implementering - Udrulning - Post mortem	0,22 (2/9) Faser omfattet: - Implementering - Udrulning

Jævn overgang	Ja	Ja	Ja	-
Gnidningsfri overgang	Nej, det er et gap mellem analyse og udvikling.	Nej, gap mellem den indledende pre-game fase og til udviklingen starter.	Nej, gap mellem planlægning og det første trin i udviklingsfasen.	-
Udviklings stil	Iterativ	Iterativ	Trinvis	Flow
Artefakter				
Tilstrækkelige produkter	0,625 (5/8) XP producerer artefakter ifm.: - Analyse - Kravspecifikation - Design - Dokumentation - Test	0,5 (4/8) Scrum producerer artefakter ifm.: - Analyse - Kravspecifikation - Design - Dokumentation	0,75 (6/8) Crystal producerer artefakter ifm.: - Analyse - Kravspecifikation - Design - Dokumentation - Test - Træning	0,125 (1/8) Kanban producerer artefakter ifm.: - Kravspecifikation
Modellering	Nej	Nej	Nej	Nej
Sammenhæng	Medium	Medium	Medium	-
Håndgribelighed/Synlighed/Test	Medium	Medium	Medium	Medium
Opfattelse	Funktionel	Funktionel	Funktionel	Funktionel
Abstraktionsniveau	Problem/løsning/implementering	Problem/Løsning/implementering	Problem/Løsning/Implementering	Problem/Løsning/Implementering
Standarder	Ja (Kodestandarder)	Nej	Nej	Nej
Krav				
Krav indsamling	Kunden skriver user stories som så bliver revideret i starten af hver iteration.	Product Owner har det overordnede ansvar for Backlog'en, samt prioritering af denne. Derudover indgår Kunden og Scrum Teamet også i dialogen omkring Backlog'en og hvordan de enkelte stories er skrevet.	En af de definerede roller skal tage rollen som forretnings ekspert. Men det er op til teamet selv at definere hvem og hvordan krav bliver indsamlet.	Ikke eksplicit defineret.
Kravspecifikations format	User story	User story	Use cases	Ikke eksplicit defineret. Det er op til projektet at vælge format og hvordan krav indsamles.
Proces baseret på funktionelle/non-funktionelle krav	Udviklingsprocessen er centreret omkring kravsspecifikatione	Udviklingsprocessen er centreret omkring den prioriterede	Udviklingsprocessen er centreret omkring krav dokumentet.	Udviklingsprocessen er centreret omkring en backlog.

	n som kunden har prioriteret.	backlog.		
Non-funktionelle krav	Skrevet i user stories	-	-	-
Sporbarhed	Ja (user stories)	Ja (user stories)	Ja (use cases)	-
Ændring af krav	Ja (user stories bliver opdateret i begyndelsen af hver iteration, og det er med muligt at komme med ændringer)	Ja (Product Owner er den overordnede ansvarlige for Backlog'en. Kravene kan dermed ændres i starten af et sprint, til fx et Sprint planning møde, hvor prioriteringen kan ændrer sig eller nye krav kan komme til.)	Ja (i starten af hvert udviklings trin vil der være planlægning af den kommende iteration, her er der mulig for ændring af krav)	Ja (Krav kan ændres i hele processen under hensyntagen til WIP. Hvis en udvikler skal lave en ændring kan det altså således først blive når han er helt færdige med igangværende opgave)
Prioritering af krav	Forretnings værdi. Kunden bestemmer prioriteringen.	Product Owner bestemmer prioriteringen - som regel ud fra forretnings værdi.	Forretnings værdi.	-
Generelle features				
Størrelse/Kompleksitet. Konkrete faser, roller, produkter og praktikker benyttet af de forskellige metoder i Tabel 6 i rapporten.	Faser: 6 Roller: 9 Produkter: 4 Praktikker 13 I alt: 32	Faser: 3 Roller: 5 Produkter: 4 Praktikker: 6 I alt: 18	Faser: 3 Roller: 4 Produkter: 5 Praktikker: 8 I alt: 20	Faser: 0 Roller: 0 Produkter: 1 Praktikker: 9 I alt: 10
Fuldstændighed Vurderet ud fra værdierne: ”Definition fuldstændighed” + ”faser i udviklingsprocessen” + ”tiltrækkelige produkter”	0,72 Udregning: (0,875 + 0,66 + 0,625)/3	0,64 Udregning: (0,875 + 0,55 + 0,5)/3	0,73 Udregning: (0,875 + 0,55 + 0,75)/3	0,28 Udregning: (0,5 + 0,22 + 0,125)/3
Praktisk anvendelse	Høj	Høj	Medium	Høj
Praktisk mulig	Høj	Høj	Høj	Høj

Modellerings sprog	**XP**	**Scrum**	**Crystal**	**Kanban**
Kriterium	-	**-**	**-**	**-**
Modelleringssprog	-	-	-	-
Nemt at lære og bruge	-	-	-	-
Styrken af sproget	-	-	-	-
Håndtering af inkonsistens	-	-	-	-
Styring af kompleksitet	-	-	-	-

Agilitet	**XP**	**Scrum**	**Crystal**	**Kanban**
Kriterium				
Hastighed	1/7 (iterationer på 1 uge)	1/7-28 (iterationer på 1 uge - 1 måned)	1/28 - 84 (iterationer på 1 måned - 3 måneder)	Kanban foreskriver ikke noget konkret tidsinterval en iteration bør løbe over. Dog anbefaler metoden at der leveres med korte mellemrum for at skabe tillid til kunden.
Bæredygtighed	Pair programming, Test-first programming,	Sprint og release burndown charts (velocity)	Review, Tracking	Focus on Quality, Measure and Manage Flow
Individer og samarbejde	1. Sit Together 2. Whole Team 3. Pair Programming	1. Scrum teams 2. Sprint planning 3. Daily Scrum	1. Holistic Diversity Strategy 2. Parallelism 3. User viewings	-
Velfungerende software	1. Test-First proramming 2. Continuous integration 3. Ten-Minute Build 4. Pair Programming 5. Weekly Cycle	1. Sprint 2. Sprint Review	1. Review 2. Tracking 3. User viewings	1. Focus on Quality 2. Reduce WIP
Samarbejde med kunden	1. Sit Together 2. Whole Team 3. Stories 4. Weekly Cycle	1. Product Backlog 2. Sprint Backlog 3. Sprint	1. Staging 2. Review 3. User viewings	1. Prioritize 2. Deliver Often

		planning		
Håndtering af forandringer	1. Weekly Cycle 2. Slack	1. Sprint review 2. Sprint retrospective 3. Daily Scrum	1. Workshops 2. Methodology-tuning technique	1. Measure and Manage Flow 2. Visualize Workflow
Lethed og simplicitet	1/32	1/18	1/20	1/10
Teknisk kvalitet	Test-first programming, Pair programming	-	-	Focus on Quality
Aktiv bruger involvering	Kunden er med til at skrive user stories i starten af hver iteration. Derudover anbefales det at kunnen sidder i samme lokale som udviklerne.	Kunden er med til at skrive user stories. Derudover er kunden i tæt dialog med udviklerne for at afstemme forventninger.	Brugeren vil få præsenteret systemet i løbet af iterationerne ved User viewings	Hvis der er overskud i WIP, kan de resterende opgaver prioriteres ved hjælp fra kunden.

Usage		XP	Scrum	Crystal	Kanban
Applikations scope					
Projekt	Størrelse	Små, Medium, Store	Små, Medium, Store	Små, Medium, Store	-
	Domæne	-		-	-
	Kultur	-	-	-	-
	Dynamik	-	-	-	-
	Kompleksitet	-	-	-	-
	Kritikalitet (tab pga. fejl)	-	-	Skønsmæssige – Essentiel finansiering	-
Udviklings team	Størrelse	<10 og flere teams	<10 og flere teams	<10 og flere teams	-
	Uddannelse	-	-	-	-
	Erfaring (i software udvikling)	Høj	-	Mindre erfarende kan mixes med mere erfarende.	-
	Erfaring indenfor domæne	Høj	-		-
	Erfaring i udviklingssprog	Høj	-	-	-
Ergonomi	Fysisk layout	Sammenstillet, distribueret	Sammenstillet, distribueret,	Sammenstillet	-
Geografi	Antal lokationer	-	-	-	-
Teknisk	Programmerings sprog	-	-	-	-
	Programmerings stil	-	-	-	-
	Abstraktions teknikker	-	-	-	-
	Obligatoriske udviklings værktøjer	-	-	Compiler	-
	Test og debug metoder	Test-First programming	-	Regressions test,	-
Ledelse	Ledelses team størrelse	-	-	-	-
	Ledelses erfaring	-	-	-	-
	Ressource allokerings metode	-	-	-	-
	Forretnings kultur	Samarbejdende	Samarbejdende	-	-
	Kunde samarbejdsmetode	Sit Together	-	-	-
	Kvantitative målinger	-	Burndown grafer	-	-
Paraply aktiviteter					

Projektledelse	0,75 (3/4) Praktikker der supportere projektledelsen: Se Tabel 11	1.0 (4/4) Praktikker der supportere projektledelsen: Se Tabel 11	1.0 (4/4) Praktikker der supportere projektledelsen: Se Tabel 11	0,75 (3/4) Praktikker der supportere projektledelsen: Se Tabel 11
Team ledelse	-	-	-	-
Kvalitets kontrol	Pair programming, Continuous integration, Test-First programming, Sit Together	Sprint review	User viewings Review	Focus on Quality, Limit WIP
Risiko ledelse	-	-	-	-
Method tailoring				
Tilpasning og customizing	Ja	Ja,	Ja	Ja,
Fleksibilitet	Ja, tilpasning af teams til projektet, og evt. opdeling af teams.	Ja, processer som fx sprintlængde er op til projektet selv at definere.	Ja, flere processer er op til det enkelte projekt selv at specificerer, bl.a. iterations længden.	Ja, flere processer er op til det enkelte projekt selv at specificerer, fx kapaciteten af WIP som også kan tilpasses
Skalerbarhed	Ja,	Ja,	Ja,	Ja,
Udvidelsesmuligheder	Ja,	Ja,	Ja,	Ja
Integration med andre metoder	Ikke nævnt	Muligt	Muligt	Muligt
Dokumenter				
Toturials og trænings dokumenter	Ja	Ja	Ja	Ja
Emperisk bevis	Ja	Ja	Ja	Ja

Cross-context	**XP**	**Scrum**	**Crystal**	**Kanban**
Hastighed	1 uge	1 uge – 1 måned	1 måned – 3/4 måneder	-
Brugbarhed	Vurderes ud fra et projekt der er benyttet på	Vurderes ud fra et projekt der er benyttet på	Vurderes ud fra et projekt der er benyttet på	Vurderes ud fra et projekt der er benyttet på
Fuldstændighed Vurderet ud fra: ”Projektledelses fuldstændighed” + ”Proces fuldstændighed”	0,8125 Udregning: (0,75 + 0,875)/2	0,9375 Udregning: (1,0 + 0,875)/2	0,9375 Udregning: (1,0 + 0,875)/2	0,625 Udregning: (0,75 + 0,5)/2
Status	Aktiv	Aktiv	Aktiv	Aktiv
Udviklingsproces	Implicit	Implicit	Eksplicit	Implicit
Modellerings sprog	Nej	Nej	Nej	Nej
Begrænsninger	Virksomhedskulturen, Teamstørrelse	Teamstørrelse	Projektet skal være placeret i samme bygning for at metoden kan benyttes.	

Appendix C

Oversigt over virksomheder

Hovedformål for de undersøgte projekter:

P1: Udvikling af nyt koncern intranet

P2: Udvikling af ny system til brug i banker

P3: Sammenlægning af to datasystemer

P4: Udvikling af nyt webbaseret system

P5: Udvikling af nyt system til forsvars industrien

Projekt	Metode	Organisations type	Organisations størrelse	Forretnings område	Marked
P1	Scrum	Statsejet	+5000	Energi	DK og Int.
P2	Scrum	Privatejet	+5000	Bank	DK og Int
P3	Scrum	Privatejet	2500	Forsikring	DK
P4	Scrum	Privatejet	90	Pension	DK
P5	Scrum	Privatejet	450	Produktudvikling	Int og Int.

Appendix D

Uddybende analyse af de fem undersøgte projekter

Ligesom med resultaterne fra afsnit 5.1, vil resultaterne blive delt op i de 5 definerede hovedområder fra CEFAM, og de væsentligste observationer vil blive præsenteret.

De enkelte elementer i frameworket vil ikke blive forklaret igen, da det er de samme, som er vurderet tidligere i afsnit 5.1. Derfor henvises der blot til dette afsnit, i tilfælde af tvivl.

Proces

Ud fra resultaterne i Appendix E, er der opstillet følgende tabel som opsummerer de væsentligste elementer fra proces evalueringen af de fem projekter.

	P1	P2	P3	P4	P5
Definition	0,75	1,0	0,625	0,875	0,875
Faser	0,55	0,55	0,44	0,33	0,66
Artefakter	0,875	1,0	0,75	0,75	1,0
Krav	Workshops	Projekt	Workshops	Eksternt	Fast scope
Generelle features	Faser: 5 Roller: 4 Produkter: 4 Praktikker: 8 I alt: 21	Faser: 5 Roller: 3 Produkter: 4 Praktikker: 4 I alt: 16	Faser: 4 Roller: 5 Produkter: 4 Praktikker: 6 I alt: 19	Faser: 3 Roller: 5 Produkter: 5 Praktikker: 7 I alt: 20	Faser: 6 Roller: 4 Produkter: 4 Praktikker: 7 i alt: 21

Tabel 17 Proces

Projekt 2 (P2) har den mest fuldstændige *definition* af processen. Projektteamet definerede processen, inden arbejdet gik i gang, og det resulterede i en omfattende definition af processer. P1 brugte de første par Sprints på at sætte rammerne for projektet, og processerne blev derfor defineret lidt mere hen ad vejen efter dialog med udviklerne. Det resulterede i en medium værdi af definition af processerne.
P3 havde samme fremgangsmåde som P1, som også først blev defineret hen af vejen, og resulterede i en middel definition af processerne. P4 og P5 havde klart defineret processen fra start af projektet. P4 var forholdsvis uerfarne i brugen af Scrum, og benyttede sig derfor af en Scrum coach til at sikre at processen var klart defineret fra start. P5 havde også processen klart defineret på forhånd, og begge projekter har dermed også nogle høje værdier for definitionen af processen. De kvantitative værdier, kan dog heller ikke i dette tilfælde, sammenlignes pga. de giver arbitrære resultater. Fx giver P4 og P5 de samme værdier, hvilket dog ikke nødvendigvis betyder de har definitioner for de samme elementer, og kan derfor ikke sammenlignes.

Alle projekterne ligger på forholdsvis lave niveauer af omfattede *faser* i udviklingsprocessen. Her er der vurderet ud fra aktiviteterne i udviklingsprocessen, hvilke faser der er omfattet af udviklingsprocessen, selvom det ikke er klart defineret fra projektets side. Der er tre faser, der går igen i alle projekterne, nemlig implementering, test og udrulning. Som nævnt i forrige afsnit vil den kvantitative værdi i dette kriterium ikke kunne bruges til en sammenligning, hvilket er diskuteret i afsnit 6.1.

Både P2 og P5 producerer *artefakter* forbundet med de 8 udviklings aktiviteter; Analyse, Design, Kravspecifikation, Modellering, Dokumentation, Test, Træning, Udrulning. Denne kvantitative værdi vil igen ikke kunne bruges til sammenligning, hvilket er yderligere diskuteret i diskussions afsnittet.

Krav er lidt forskelligt håndteret i de fem projekter. I P1 bliver der blev afholdt workshops vedr. *user stories* lige inden Sprint planlægnings mødet. Deltagerne her er *Scrum Master*, *Product Owner*, kunden og de udførende udviklere. Outputtet fra workshopen er opdaterede *user stories* samt et mødereferat. Dette sikrer forventnings afstemningen med kunden. I P2 havde et foregående projekt til opgave at indsamle krav. Disse blev defineret i form af *high level use-cases*, som af projektet blev omformuleret til *user-stories*. P3 har nogenlunde samme fremgangsmetode som P1. Her bliver krav også indsamlet ved workshops hvor både kunde og udviklere er til stede. En god dialog sikrer, at kravene bliver samlet til *user stories* ved disse workshops. I P4 stod et eksternt selskab for at lave kravindsamlingen, og definere kravspecifikationen. Kravspecifikationen fra det eksterne selskab blev skrevet om til *user stories* af samarbejdspartnere til projektet. P5 havde et fast *scope* for projektet, og alle krav var derfor fast defineret, som *use cases*, på forhånd inden projektet startede.

Generelle feature er uddybet i Appendix . Her kan ses helt konkret hvilke faser, roller, produkter og praktikker der er defineret for de enkelte projekter.

Modellerings Sprog

Modellerings sprog er udelukkende benyttet af P1 og P5. I P1 blev der benyttet UML til analyse og design, hvor der blev udarbejdet nogle *use-case* diagrammer, men det var ikke et krav. I P5 blev UML også brugt, men igen var det ikke et krav, og det blev udelukkende brugt på et ad hoc basis, når der var et behov for det fra udviklernes side.

Agilitet

Ud fra resultaterne i Appendix E, er der opstillet følgende tabel som opsummerer de væsentligste elementer fra proces evalueringen af de fem projekter.

	P1	**P2**	**P3**	**P4**	**P5**
Hastighed	Iterationer på 2 uger	Iterationer på 3 uger	Iterationer på 2 uger	Itererationer på 2 uger	Iterationer på 4 uger
Individer og samarbejde	1. *Sprint planning* 2. *Pre-sprint planning* 3. *Daily Scrum* 4. *Scrum team*	1. *Sprint planning* 2. *Scrum teams* 3. *Daily Scrum*	1. *Sprint planning* 2. *Scrum team* 3. *Daily Scrum*	1. *Sprint planning* 2. *Scrum teams* 3. *Daily Scrum*	1. *Sprint planning* 2. *Scrum teams* 3. *Daily Scrum*
Velfungerende software	1. *Sprint* 2. *Pair Programming*	1. *Sprint*	1. *Sprint* 2. *Pair programming* 3. *Test-First Programming*	1. *Sprint* 2. *Test-First Programming*	1. *Sprint* 2. *Pair programming* 3. *Reduce WIP*
Samarbejde med kunden	1. *Product backlog* 2. *Sprint backlog* 3. *Sprint planning* 4. *Pre-sprint planning* 5. *Sit Together*	1. *Stories* 2. *Product Backlog* 3. *Sprint Backlog* 4. *Sprint planning*	1. *Product Backlog* 2. *Sprint Backlog* 3. *Sprint planning*	1. *Product Backlog* 2. *Sprint Backlog* 3. *Sit Together* 4. *Sprint planning*	1. *Product Backlog* 2. *Sprint Backlog* 3. *Sprint planning*
Håndtering af forandringer	1. *Sprint retrospective*	1. *Sprint retrospective*	1. *Sprint retrospective*	1. *Sprint review* 2. *Sprint*	1. *Sprint review* 2. *Sprint*

	2. *Daily Scrum*	2. *Daily Scrum*	2. *Daily Scrum*	*retrospective* 3. *Daily Scrum*	*retrospective* 3. *Daily Scrum* 4. *Visualize Workflow*
Lethed og simplicitet	1/21	1/16	1/19	1/20	1/21

Tabel 18 Agilitet

P1, P3 og P4 benytter sig alle sammen af iterationer på 2 ugers varighed, mens P2 har 3 ugers iterationer. P5 har iterationer på 4 uger, men planlægger kun 2 uger frem i tiden. Dvs. at de har *Sprint planning* hver 2. uge, og *Sprint review* og *retrospective* hver 4. uge. De afleverer altså færdig kode hver 4. uge.

Ligesom i teori afsnittet er projekterne her blevet evalueret ud fra det agile manifest, for at kunne sammenligne hvilke agile praktikker projekterne benytter sig af.

Alle projekterne benytter sig af *Sprint planning*, *Daily Scrum* og har et eller flere Scrum teams. P1 har tilføjet et *Pre-sprint planning* møde, som ellers ikke er nævnt i Scrum, hvor kunden og *Scrum Masteren* startede den indledende prioritering af *Product Backlog'en*.

I praktikkerne der understøtter "*Velfungerende software frem for omfattende dokumentation*" er Sprint den eneste fællesnævner i alle projekterne. Dog var der flere af projekterne, der har tilføjet flere elementer fra andre metoder. P1, P3 og P5 benyttede sig alle af en form for *pair programming*. P1 lavede kode *reviews*, hvor to udviklere gennemgik hver færdigudviklet *user story*, mens P5 benyttede sig af *inspections* hvor to udviklere gennemgik tankerne bag koden, og sikrede den overholdt arkitekturen. P5 benyttede i en kort overgang Kanban praktikken *Reduce WIP*, dette vil blive diskuteret yderligere i senere afsnit.

Fællesnævnerne i praktikkerne der understøtter "*Samarbejde med kunden*" er *Product backlog*, *Sprint Backlog* og *Sprint planning*. Alle fem projekter benytter sig af disse praktikker som fremmer samarbejdet med kunden. Derudover benyttede P1 og P4 sig af *Sit Together*, for at sikre kunden altid var tæt på udviklerne, og eventuelle afklaringer hurtigt kunne blive løst.

Håndtering af forandringer viser, at alle projekterne benytter sig af *Sprint Retrospective* og *Daily Scrum*. P4 og P5 tilføjer dog også, et *Sprint Review* for at sikre der er opnået det ønskede resultat i hvert Sprint.

Resultaterne for *Lethed og simplicitet* der viser præcis hvilke faser, roller, produkter og praktikker kan findes i Appendix C. I og med at alle projekterne benytter sig af samme metode, ligger resultaterne også forholdsvist tæt.

Brugbarhed
Se afsnit 5.2 i rapporten

Cross-context
Ud fra resultaterne i Appendix D, er der opstillet følgende tabel som opsummerer de væsentligste elementer fra cross-context evalueringen af de fem projekter.

	P1	**P2**	**P3**	**P4**	**P5**
Hastighed	Iterationer på 2 uger	Iterationer på 3 uger	Iterationer på 2 uger	Itererationer på 2 uger	Iterationer på 4 uger
Brugbarhed	Høj grad	Høj grad	Høj grad	Høj grad	Høj grad

Fuldstændighed	0,87	1,0	0,81	0,94	0,94
Udviklingsproces	Implicit	Implicit	Implicit	Explicit	Explicit
Begrænsninger	-	-	-	-	Ja

Tabel 19 Cross-context

Hastighed er vurderet ud fra længden af iterationer som projekterne benyttede. Her ses det at P1, P3 og P4 alle benytter sig af Sprint af 2 ugers varighed, mens P2 og P5 benyttede sig af hhv. 3 og 4 uger.

Brugbarhed er en subjektiv vurdering fra testpersonerne der vurderer i hvor høj grad processen har været brugbar for dem. Alle testpersonerne var enige i at Scrum processen i høj grad var brugbare for deres projekter.

Fuldstændighed er en definition af den generelle proces fuldstændighed og projektledelses fuldstændighed. Ligesom i resultaterne fra analysen af teorien, vil den kvantitative værdi her ikke kunne bruges til en sammenligning.

Udviklingsproces er resultaterne fra om metoderne har en eksplicit eller implicit måde at forklare selve udviklingsprocessen på. P1, P2 og P3 definerede alle processen lidt hen ad vejen og havde altså ikke nogen klar definition fra start af. P4 benyttede sig af en Scrum coach, som klart definerede processen for dem inden projektet gik i gang. P5 have også processen klart defineret inden projektet startede.

Begrænsninger er de generelle begrænsninger som der er opsat for nogle af metoderne. Dette er en subjektiv vurdering fra testpersonerne omhandlende om de fandt nogle begrænsninger i brugen af Scrum metoden. Her nævner P5, at de den begrænsning at det kunne være svært at se det store perspektiv. De korte iterationer gjorde det til tider svært at se det store perspektiv, hvilket kan være med til at miste overblikket.

Appendix E

Samlede resultater fra interviews

	P1	P2	P3	P4	P5
Proces					
Definition					
Eksplicithed og entydighed	Nej, de første par sprints blev brugt til at sætte rammerne for projektet. Dvs. udviklingsprocessen blev defineret i løbet af de første par sprints via dialog ml. udviklerne.	Ja, Projektteamet definerede selv procesen inden den gik i gang.	Nej, det er op til projekt teamet at definere hvilken proces man vil benytte, derfor blev besluttet og justeret hen af vejen.	Ja, klart defineret af Scrum coach inden projektet gik i gang.	Ja, processen blev klart defineret inden projektet startede.
Rationale	Nej	Nej	Nej	Ja	Ja,
Fuldstændighed	0,75 (6/8) P1 har definitioner for: - Udviklingscyklus - Roller - Aktiviteter - Produkter - Praktikker - Paraply aktiviteter	1,0 (8/8) P2 har en komplet definition af alle elementerne i processen.	0,625 (5/8) P3 har definitioner for: - Udviklingscyklus - Roller - Aktiviteter - Produkter - Praktikker - Paraply aktiviteter	0,875 (7/8) P4 har definitioner for; - Udviklingscyklus - Roller - Aktiviteter - Produkter - Praktikker - Regler - Paraply aktiviteter	0,875 (7/8) P5 har definitioner for; - Udviklingscyklus - Roller - Aktiviteter - Produkter - Praktikker - Regler - Paraply aktiviteter
Faser					
Faser i udviklingsprocessen	0,55 (5/9) Faser omfattet: - Analyse - Implementering - Test - Udrulning -	0,55 (5/9) Faser omfattet: - Analyse - Design - Implementering - Test - Udrulning	0,44 (4/9) Faser omfattet: - Implementering - Test - Udrulning - Vedligeholdelse	0,33 (3/9) Faser omfattet: - Implementering - Test - Udrulning Kommentar: Analyse og	0,66 (6/9) Faser omfattet: - Analyse - Design - Implementering - Test - Udrulning -

	Vedligeholdelse Kommentar: Analyse vil indgå efter behov i et sprint. Typisk i forbindelse med gennemgang af user stories med forretningen.		Kommentar: Analyse og design indgår ad hoc i et sprint når det er behov for det. Der vil være support af projektet 2-3 måneder efter det er afsluttet, men denne fase starter først når projektet er afsluttet.	design blev lavet inden de første iterationer startede. Det blev dog brugt i nogle tilfælde i sprintsne til yderligere forklaringer, men kun på et ad hoc basis.	Vedligeholdelse Kommentar: Support og Post mortem faserne vil først indtræde efter projektet er afsluttet i og med det er et nyt produkt. Support vil køre så længe produktet er på markedet, mens post mortem først vil indtræde når produktet udfases af produktporteføl gen.
Jævn overgang	Ja (workshops)	-	Ja (Sprint planlægning og sprint retrospective)	Ja (Sprint planlægning og sprint retrospective)	Ja, faserne er datospecifikke.
Gnidningsfri overgang	Nej (tæt dialog m. forretning)	-	Nej	Nej	Ja
Udviklings stil	Iterativ	Iterativ	Iterativ	Iterativ	Iterativ
Artefakter					
Tilstrækkelige produkter	0,875 (7/8) Projektet producerer artefakter ifbm.: - Analyse - Kravspecifikatio n - Design - Dokumentation - Test - Træning - Udrulning	1,0 (8/8) Projektet producerer artefakter fundet med alle 8 udviklings aktiviteter.	0,75 (6/8) Projektet producerer artefakter ifbm.: - Analyse - Kravspecifikatio n - Dokumentation - Test - Træning - Udrulning	0,75 (6/8) Projektet producerer artefakter ifbm. : - Kravspecifikatio n - Design - Dokumentation - Test - Træning - Udrulning	1,0 (8/8) Projektet producer artefakter forbundet med alle 8 udviklings aktiviteter.
Modellering	Nej	Ja, UML benyttes.	Nej	Nej	Ja, modellering vil omfatte

					analyse af fx prototyper og GUI.
Sammenhæng	Medium	Medium	Medium	Medium	Medium
Håndgribelighed/Synlighed/Test	Høj	Medium	Høj	Medium	Høj. Nogle af produkterne der produceres vil blive brugt som en del af det endelige produkt, fx Brugermanual.
Opfattelse	Funktionel	Funktionel	Funktionel	Funktionel	Funktionel
Abstraktionsniveau	Problem/løsning/Implementation	Problem/Løsning/Implementation	Problem/Løsning/Implementation	Problem/Løsning/Implementation	Problem/Løsning/Implementering
Standarder	Ja (templates til dokumentation af projektet)	Nej	Nej	Ja (Kodestandarder)	Nej
Krav					
Krav indsamling	Der blev afholdt workshops vedr. user stories. Deltagere: Scrum Master, Product Owner, Forretning, udførende udviklere. Output fra workshop er opdaterede user stories samt mødereferat. Dette sikrer forventnings afstemningen med forretningen.	Et foregående forretningsprojekt havde til opgave at indsamle krav. Disse blev defineret i form af high level use-cases, som af projektet blev omformuleret til user-stories.	Krav bliver samlet ved workshops hvor både forretning og udviklere er til stede. Dialog mellem disse bliver samlet til user stories.	Eksternt selskab stod for at lave kravindsamlingen. Kravspecifikationen fra det eksterne selskab blev skrevet om til user stories af samarbejdspartnere til projektet.	Kravene var fast definerede på forhånd inden projektet startede.
Kravspecifikations format	User story	User story	User story	User story	Use case
Proces baseret på funktionelle/non-	Ja, prioriteret backlog.	Ja, prioriteret backlog.	Ja, prioriteret backlog danner bacgrund for sprint	Ja, user stories blev prioriteret af forretningen, estimeret af	Ja, der bliver skrevet test til kravene for at sikre at

funktionelle krav			backlogen.	udviklerne og delt op i sprint.	implementering en understøtter kravene.
Non-funktionelle krav	Ikke direkte håndteret. Fx blev performance måling udført som en sideløbende aktivitet.	En arkitektur gruppe definerede de non-funktionelle krav i user-stories.	Ad hoc håndteret. Nogle gange defineret i undergrupper til user stories hvilket skaber forvirring blandt udviklere. Nogle gange også beskrevet som en user story.	Non-funktionelle krav blev nedskrevet i dokumenter. Ikke nogen direkte sammenhæng med user stories.	Non-funktionelle krav blev beskrevet i test cases.
Sporbarhed	Ja (user stories, møde referater og desicion log)	Ja (dokumentation af kode)	Ja (user stories)	Ja (user stories)	Ja (use cases)
Ændring af krav	Ja (direkte dialog mellem Product Owner og Scrum Master)	Ja (ved en data change request)	Ja (ved direkte dialog mellem forretningen og Scrum Master)	Ja (ved direkte dialog)	Ja (ved dialog ml. Product Owner og udviklerne)
Prioritering af krav	Product Owner foretog prioritering i tæt dialog med Scrum Master efter forretnings værdi	Ud fra flere parametre bl.a.: Forretningsværdi og IT kompleksitet	Product Owner proriterer kravene ud fra forretningsmæs sig værdi.	Workshop efter hvert sprint hvor krav blev prioriteret af Product Owner på baggrund af forretningsmæs sig værdi.	Bruger- og forretningsmæs sig værdi
Generelle features					
Størrelse/Kompleksitet Konkrete faser, roller, produkter og praktikker benyttet af de forskellige projekter kan findes i Appendix F.	Faser: 5 Roller: 4 Produkter: 4 Praktikker: 8 I alt: 21	Faser: 5 Roller: 3 Produkter: 4 Praktikker: 4 I alt: 16	Faser: 4 Roller: 5 Produkter: 4 Praktikker: 6 I alt: 19	Faser: 3 Roller: 5 Produkter: 5 Praktikker: 7 I alt: 20	Faser: 6 Roller: 4 Produkter: 4 Praktikker: 7 i alt: 21 Kommentar: I en kort periode hvor projektet udelukkende kørte test blev Kanban indført. Her benyttede man sig

					udelukkende af "Reduce Work-in-Progress" og "Measure and Manage Flow".
Fuldstændighed Udregnet ud fra: ”Definition fuldstændighed” + ”faser i udviklingsprocessen” + ”Tiltrækkelige produkter”	0,72 Udregning: (0,75 + 0,55 + 0,875)/3	0,85 Udregning: (1,0 + 0,55 + 1,0)/3	0,6 Udregning: (0,625 + 0,44 + 0,75)/3	0,65 Udregning: (0,875 + 0,33 + 0,75)/3	0,84 Udregning: (0,875 + 0,66 + 1,0)/3
Praktisk anvendelse	Høj	Høj	Høj	Høj	Høj
Praktisk mulig	Høj	Høj	Høj	Høj	Høj

Modellerings sprog	P1	P2	P3	P4	P5
Kriterium					
Modelleringssprog	-	UML, blev benyttet til analyse og design.	-	-	UML, men ikke et krav og blev kun brugt når der var behov for det for udviklernes side. .
Nemt at lære og bruge	-	-	-	-	
Styrken af sproget	-	-	-	-	
Håndtering af inkonsistens	-	-	-	-	
Styring af kompleksitet	-	-	-	-	

Agilitet	**P1**	**P2**	**P3**	**P4**	**P5**
Kriterium					
Hastighed	1/14 (iterationer på 2 uger)	1/21 (iterationer på 3 uger)	1/14 (iterationer på 2 uger)	1/14 (iterationer på 2 uger)	1/28 (iterationer på 4 uger, men med Sprint planning hver 2. uge, og Sprint review og retrospective hver 4. uge)
Individer og samarbejde	1. *Sprint planning* 2. *Pre-sprint planning* 3. *Daily Scrum* 4. *Scrum team*	1. *Sprint planning* 2. *Scrum teams* 3. *Daily Scrum*	1. *Sprint planning* 2. *Scrum team* 3. *Daily Scrum*	1. *Sprint planning* 2. *Scrum teams* 3. *Daily Scrum*	1. *Sprint planning* 2. *Scrum teams* 3. *Daily Scrum*
Velfungerende software	1. *Sprint* 2. *Pair Programming*	1. *Sprint*	1. *Sprint* 2. *Pair programming* 3. *Test-First Programming*	1. *Sprint* 2. *Test-First Programming*	1. *Sprint* 2. *Pair programming* 3. *Reduce WIP*
Samarbejde med kunden	1. *Product backlog* 2. *Sprint backlog* 3. *Sprint planning* 4. *Pre-sprint planning* 5. *Sit Together*	1. *Stories* 2. *Product Backlog* 3. *Sprint Backlog* 4. *Sprint planning*	1. *Product Backlog* 2. *Sprint Backlog* 3. *Sprint planning*	1. *Product Backlog* 2. *Sprint Backlog* 3. *Sit Together* 4. *Sprint planning*	1. *Product Backlog* 2. *Sprint Backlog* 3. *Sprint planning*
Håndtering af forandringer	1. *Sprint retrospective* 2. *Daily Scrum*	1. *Sprint retrospective* 2. *Daily Scrum*	1. *Sprint retrospective* 2. *Daily Scrum*	1. *Sprint review* 2. *Sprint retrospective* 3. *Daily Scrum*	1. *Sprint review* 2. *Sprint retrospective* 3. *Daily Scrum* 4. *Visualize Workflow*
Lethed og simplicitet	1/21	1/16	1/19	1/20	1/21
Teknisk kvalitet	Kode review for hver user story	QA funktion sikrede kvalitets kontrollen	Løbende test og par programmering.	Løbende test, kodereview fra eksternt bureau sikrer at alle kodestandarder er overholdt.	Løbende test, Inspections(gennemgang af kode)
Aktiv bruger involvering	Product Owner og Forretningen er samme person. Forretningen er med til definering af user stories ved workshops.	Product Owner ser demo af systemet i slutningen af hvert sprint.	Product Owner og forretningen er samme person, som er i tæt dialog med alle interessenter.	Product Owner var ansvarlig for prioritering af backlog og deltog i dialogerne i sprint planlægningen.	Kunden og Product Owner er samme person. Derudover vil slut kunderne teste produktet og udtale sig omkring det.

Usage		P1	P2	P3	P4	P5
Applikations scope						
Projekt	Størrelse	9 mdr.	17 mdr.	18 mdr.	7 mdr.	15 mdr.
	Domæne	Teknologi opdatering/Ny udvikling	Ny udvikling	Data integration	Ny udvikling	Produktudvikling
	Kultur	60%	10%	30%	50%	70%
	Dynamik	40%	10%	30%	10%	20%
	Kompleksitet	Medium	Medium	Medium	Medium	Høj
	Kritikalitet (tab pga. fejl)	Skønsmæssige/ essentielle midler	Essentielle midler	Skønsmæssige/ essentielle midler	Essentielle midler	Liv
Udviklings team	Størrelse	10	80	7	15	28
	Uddannelse	Level 2&3 personer	Level 2&3 personer	Level 2&3 personer	Level 2&3 personer	Level 2&3 personer
	Erfaring (i software udvikling)	5-7 år	8-x	5-7	-	5-7
	Erfaring indenfor domæne	2-4 år	8-x	5-7	-	5-7
	Erfaring i udviklingssprog	2-4 år	8-x	5-7	-	2-5
Ergonomi	Fysisk layout	Sammenstillet	Distribueret	Sammenstillet	Distribueret	Distribueret
Geografi	Antal lokationer	2	5	1	3	2
Teknisk	Programmerings sprog	C#	Flere	PI1, Java	.Net	Java
	Programmerings stil	Simpel	Kompleks	Kompleks	Kompleks	Kompleks
	Abstraktions teknikker	Object-orienteret	Object-orienteret	Traditionel	-	Object-orienteret
	Obligatoriske udviklings værktøjer	Vision studio 2008, Team foundation 2008	-	-	-	-
	Test og debug metoder	HP QC 2009	-	HP QC	-	Idea og TestLink
	Ledelses team størrelse	Lille	Medium	Medium	Medium	Lille

Ledelse	Ledelses erfaring	Medium	Høj	Høj	Høj	Høj
	Ressource allokerings metode	Planning Game	-	-	-	Planning Game
	Forretnings kultur	Meget samarbejdende	Samarbejdende	Samarbejdende	Samarbejdende	Meget samarbejdende
	Kunde samarbejds metode	Kundetilstedeværelse	Kundetilstedeværelse	Kundetilstedeværelse	Kundetilstedeværelse	Kundetilstedeværelse
	Kvantitative målinger	Velocity graf, Burndown graf	Velocity graf	Velocity graf	Velocity graf	Velocity grafer
Paraply aktiviteter						
Projektledelse		1.0 (4/4) Aktiviteter der supportere PL: Se oversigt i Appendix D	0,75 (3/4) Aktiviteter der supportere PL: Se oversigt i Appendix D	0,75 (3/4) Aktiviteter der supportere PL: Se oversigt i Appendix D	1.0 (4/4) Aktiviteter der supportere PL: Se oversigt i Appendix D	1.0 (4/4) Aktiviteter der supportere PL: Se oversigt i Appendix D
Team ledelse		Parprogrammering	Parprogrammering blev foreslået, men det blev ikke brugt i praksis.	-	-	-
Kvalitets kontrol		Kode reviews	QA funktion stod for kvalitetskontrollen	Løbende test fra forretningen	Ekstern kode review	Insepctions
Risiko ledelse		Risikoanalyse ud fra den overordnede projektmodel.	-	Generelle praktikker fra overordnet projektmodel der understøttet risikoanalyse	-	Praktikker der defineret ud fra den overordnede projektmodel.
Method tailoring						
Tilpasning og customizing		Ja,	Ja	Ja	Ja,	Ja,
Fleksibilitet		Ja, processer ændret undervejs (tilføjet review proces)	Ingen processer ændret undervejs.	Ja, processer ændret undervejs (fra 1 måned til 2 ugers sprint)	Ingen processer ændret undervejs.	Ja, processer ændret undervejs (fra 3 Scrum Masters til 1)
Skalerbarhed		Ja, positiv erfaring med at bruge metoden	Ja, kan skaleres til stort projekt	Ja	-	Ja

	ved projekt med fast scope.	vha. scrum-of-scrums.			
Udvidelsesmulighed er	Ja	Ja,	Ja	-	Ja
Integration med andre metoder	Brugt og benyttet.	Brugt og benyttet	Brugt og benyttet	Brugt og benyttet	Brugt og benyttet
Dokumenter					
Toturials og trænings dokumenter	Ja	Ja	Ja	Ja	Ja
Emperisk bevis	Ja	Ja	Ja	Ja	Ja

Cross-context	**P1**	**P2**	**P3**	**P4**	**P5**
Hastighed	iterationer på 2 uger	iterationer på 3 uger	iterationer på 2 uger	iterationer på 2 uger	iterationer på 4 uger
Brugbarhed	I høj grad	I høj grad	I høj grad	I høj grad	I høj grad
Fuldstændighed Vurderet ud fra: "Projektledelses fuldstændighed" + "Proces fuldstændighed"	0,775 Udregning: (0,55 + 1,0)/2	0,65 Udregning: (0,55 + 0,75)/2	0,595 Udregning: (0,44 + 0,75)/2	0,665 Udregning: (0,33 + 1,0)/2	0,83 Udregning: (0,66 + 1,0)/2
Status	Aktiv	Aktiv	Aktiv	Aktiv	Aktiv
Udviklingsproces	Implicit	Implicit	Implicit	Explicit	Explicit
Modellerings sprog	Nej	Ja	Nej	Nej	Ja
Begrænsninger	-	-	-	-	Ja, svært at have det store overblik

Appendix F

Oversigt over projekterne

Oversigt over faser, roller, produkter og praktikker benyttet i de fem undersøgte projekter.

Projekt	P1	P2	P3	P4	P5
Faser	1. Analyse 2. Implementering 3. Test 4. Udrulning 5. Vedligeholdelse	1. Analyse 2. Design 3.Implementering 4. Test 5. Udrulning	1.Implementering 2. Test 3. Udrulning 4.Vedligeholdelse	1.Implementering 2. Test 3. Udrulning	1. Analyse 2. Design 3.Implementering 4. Test 5. Udrulning 6.Vedligeholdelse
Roller	6. Scrum Master 7. Product Owner/Forretning 8. Scrum Team 9.Styregruppe	6. Scrum Master 7. Product Owner 8. Scrum Team	5. Scrum Master 6. Product Owner 7. Scrum Team 8. Kunden 9. Ledelse	4. Scrum Master 5. Product Owner 6. Scrum Team 7. Kunden 8. Ledelse	7. Scrum Master 8. Product Owner 9. Scrum Team 10. Ledelse.
Produkter	10. Product Backlog 11. Sprint Backlog 12. Sprint burndown chart 13. QC fejlrapport	9. Story Cards 10. Product Backlog 11. Sprint Backlog 12. Sprint Burndown	10. Product Backlog 11. Sprint Backlog 12. Sprint Burndown chart 13. Release burndown chart	9. Product Backlog 10. Sprint Backlog 11. Sprint burndown chart 12. Release burndown chart 13. Code standards	11. Product Backlog 12. Sprint Backlog 13. Sprint burndown chart, 14. Release burndown chart,
Praktikker	14. Sprint 15. Sprint planning 16. Sprint retrospective 17. Daily scrum 18. Pre-sprint planning, 19.Review/pairprogramming 20.Sit Together 21. Planning game.	13. Sprint 14. Sprint planning 15. Sprint retrospective 16. Daily Scrum	14.Parprogrammering, 15. Sprint 16. Sprint planning 17. Sprint retrospective 18.Daily Scrum 19. Test-First Programming.	14. Sprint 15. Sprint planning, 16. Sprint review 17. Sprint retrospective 18. Daily Scrum, 19. Test-First programming, 20. Sit Together	15. Sprint, 16. Sprint planning, 17. Sprint review 18. Sprint retrospective 19.Daily Scrum 20. Planning Poker 21.Inspections (Pair programming)