## Unified Process (Louise)

### Hvad er UP?

Unified Process er en systemudviklingsprocess, der er udviklet i slut '90erne af Ivar Jacobsen, Grady Booch og James Rumbaught.

UP anvender UML notation, og er desuden brugt og omtalt af bl.a. Craig Larman.

UP er særligt velegnet i meget store projekter, der enten har mange mennesker involveret, eller strækker sig over længere tid.

### Unified Process i et lille projekt (Louise)

Her er projekter som vores, ikke ret velegnede til at kører UP, da planen med bl.a. projektplan og faser kan falde lidt til jorden, når man sidder i et så lille team, at det er muligt at opretholde god kommunikation med alle i gruppen.

Hvis vi kigger på projektets opsætning, er der flere steder, hvor vi i stedet ville have haft gavn af nogle af principperne fra Scrum, og fx har vi de dage vi har været på skolen, næsten dagligt haft et lille morgenmøde, hvor vi har snakket om, hvad vi lavede den foregående dag, og hvad dagen skal forløbe med.

Desuden ville et Scrumboard formentlig have bragt mere værdi i vores projekt, end vores projektplan gjorde, da vi har manglet fleksibiliteten i projektplanen, og derfor ofte fået rykket el. udsat små opgaver. Vi har desuden overset et par opgaver, der ville være husket bedre med en TODO, så de kunne være udført når gruppen samlet set var klar til det.

Der er tale om en iterativ udviklingsmetode, hvilket vil sige, at man bevæger sig i cirkler, og hele tiden holder kontakt med kunden og sørger for, at kravene er up to date.

Programmet udvikles på denne måde i mindre steps (inkrementerende), hvilket giver udviklerne mulighed for, at udnytte den viden de opnår igennem projektet.

På denne måde undgår man tidligere problemstillinger, som er kendt fra vandfaldsmodellen, når denne anvendes på større projekter, der strækker sig over længere tid.

## Faserne (Louise)

Unified Process er inddelt i fire overordnede faser: Inception, Elaboration, Construction og Transition.

Faserne er det øverste led i projektets plan. Der er forskel på, hvor længe man er i de forskellige faser. I hver dase er desuden delt ind i en el. flere Iterationer, som det er afbilledet på nedenstående billede. Der er forskelligt fra projekt til projekt, hvor mange iterationer der ligger i hver fase. I vores projekt har der fx været én Iteration i Inception (Hvilket er meget typisk) og derefter 5 iterationer i Elaboration, da vi simpelthen ikke har været klar til at gå i Construction, grundet manglende dokumentation.

[[1]](#footnote-1)

### Inception (forberedelsen) (Louise)

Inception den første fase. Denne fase er desuden også den klart korteste. I denne fase fastslås det, hvad meningen er med projektet, der udarbejdes en business case, Use Cases identificeres, risiko adresseres og man estimerer på, hvad det vil koste at lave.

Når man forlader Inception-fasen bør der altså foreligge:

* Et visionsdokument
* En Use Case model (med en liste over de aktører og Use Cases, der kan identificeres på så tidligt et tidspunkt)
* En påbegyndt Dataordbog
* En Business Case (indeholdende lidt om forretningen, et succes kriterie og en prognose af den finansielle.)
* En risikoanalyse
* En projektplan

Selve Inception fasen forløb ret planmæssigt i vores projekt og strakte sig over lidt over en dag. Vi havde dog glemt risikoanalysen, der derfor tilkom ret sent i forløbet.

Man kunne have argumenteret for, at udarbejdelse af vores primærer Use Case på Fully dressed niveau skulle ligge i denne fase, samt udvikling af prototyper, på de ukendte områder, men grundet projektets størrelse, valgte vi at lægge dette i Elaboration.

### Elaboration (etableringen)(Louise)

Elaboration er den mest kritiske fase. Det er i denne man beslutter, om vi ønsker at færdiggører projektet, eller om der er nogle problematiske områder, der gør at vi vælger at droppe projektet (dette kan både være grundet en høj risiko, der er forbundet med projektet, eller at projektets færdige omkostninger vil overstige det budget, der er estimeret i Inception. I denne fase skal al beskrivelsen ligge, desuden skal den mest væsentlige handling i programmet kodes. Når man forlader Elaboration, og bevæger sig ind i Construction, skal man være sikker på, at projektet føres til ende.

Når man forlader denne fase bør der foreligge:

* En Use Case model, der mindst er 80 % færdig
* De andre krav bør være beskrevne, her i blandt de ikke funktionelle krav.
* En fuld beskrivelse af Softwarets arkitektur
* Prototyper af nødvendige områder

Vores gruppe forlod aldrig Elaboration, da vi manglede indtil flere af de ting, vi burde have før dette skete. Fx var vores Use Cases ikke 80 % beskrevet før i den aller sidste uge, det samme gør dig gældende for krav.

### Use Case Model (Louise)

Use Cases er artifacter, der har indflydelse på mange andre artifacter. Use Case modellen er opfundet af Ivar Jacobson, Der også er en af grundlæggerne af UP, i 1986. Heraf Use Cases meget centrale rolle i UP.

Omkring 10 % af de mest kritiske Use Cases bør være beskrevet allerede i Inception, og alle Use Cases bør være identificeret og i kort form, allerede inden visionen skrives.

Når man identificerer Use Cases bør man først kigge på systemet, og derefter hvem der er aktører her er.

I vores tilfælde kunne man spørge sig selv, hvorfor det er sælgeren, og ikke kunden der er primær aktør. Svaret på dette skal findes i systemet. Det er vores sælger, der er bruger af systemet, og selve salget til kunden er derfor ikke et mål for denne. Sælgeren har til gengæld et mål, der hedder at sælge biler til kunderne, og i forbindelse med dette låne penge ud. Til dette skal han oprette et lån, som må siges at være vores mest centrale Use Case.

Der er 3 forskellige tests, der kan udføres på en Use Case, for at se, om den opfylder de basale krav for at være en Use Case.

* Boss Test

Use Casen skal være en handling, som chefen ville være tilfreds med, at hans ansatte udførte hele dagen.

* EBP Test (Elemental Business Process)

Ligesom med Boss Testen skal Use Casen tilfører virksomheden en eller anden form for værdi. Det kan fx være at godkende låneanmodninger.

* Size Test

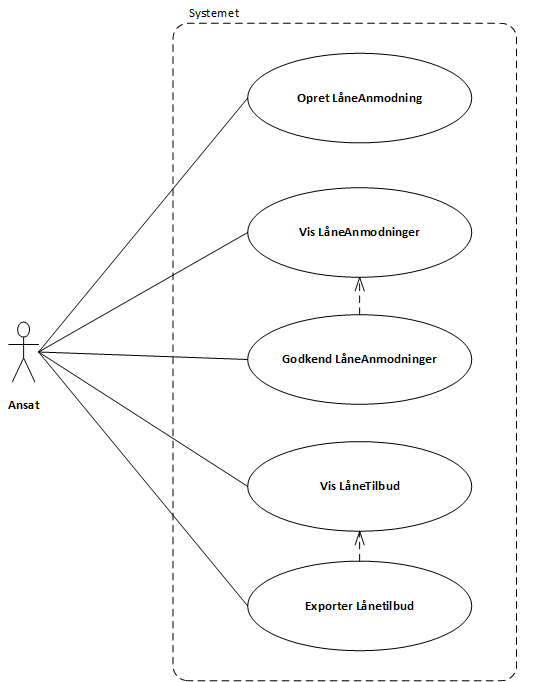
En Fully dressed Use Case vil som regelt være af en hvis størrelse, så hvis vi er i besiddelse af en Use Case, der kun indeholder et enkelt trin, vil den ikke bestå en Size Test.

* Undtagelser

Der kan være tilfælde, hvor en Use Case får lov at forblive en Use Case, på trods af, at den ikke består en test. Her kan vi se på vores "Log ind" Use case, som ikke ville bestå en Boss Test, men på grund at sin kompleksitet og vigtighed for lov at forblive.

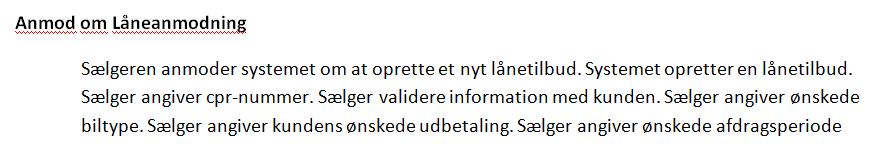
Der er både Use Case Diagrammer, Brief Use Cases, Casual Use Cases og Fully Dressed Use Cases.

**Use Case Diagrammet** er ikke en artifakt, der ikke kan stå alene, men kan bruges i en kontekst, til at skabe overblik over sammenspillet mellem aktører og handlinger. Desuden bruges de som indputs til Use Case Teksten.

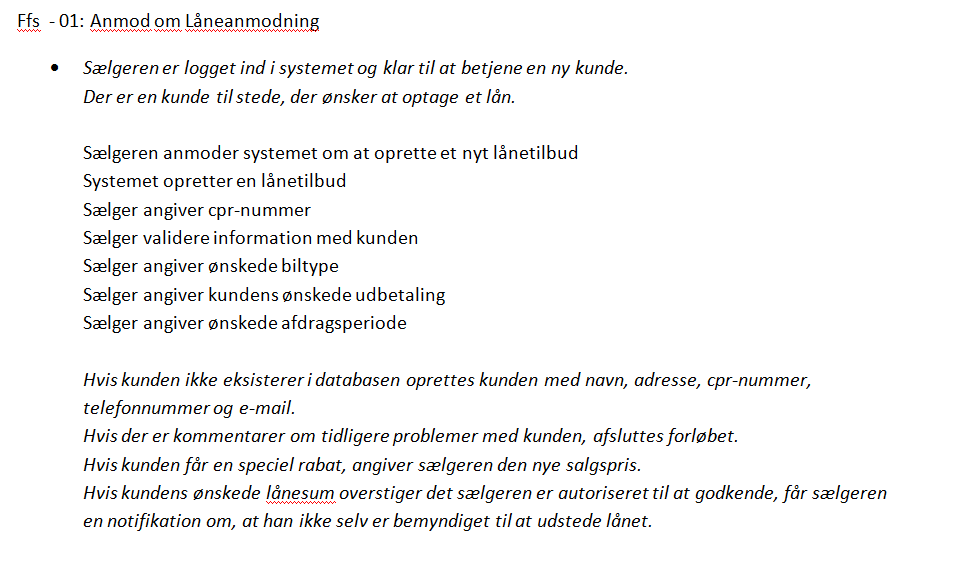


Aktøren vil indgå i Use Case teksten og de identificerede Use Cases vil være overskrifter på de nedskrevne Use Cases.

**Brief Use Cases** er det første, der skrives ned, og er egentligt bare en kort, tekstuel beskrivelse af forløbet, uden nogle afvigelser.



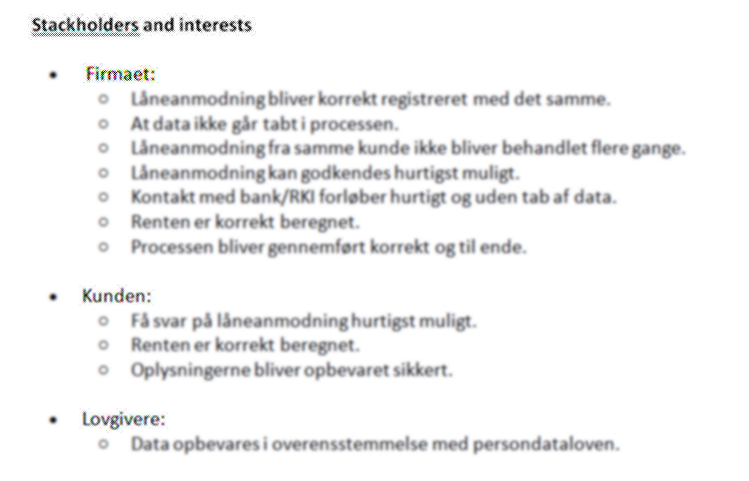
En **Casual Use Case** er på samme måde kort, men med afvigelser, og desuden mere struktureret opsat.



En **Fully Dressed Use Case** er meget længere, og indeholder alt, der kan identificeres i forbindelse med handlingen.

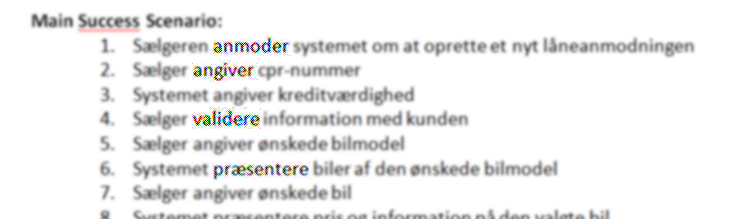
Én af de vigtige ting, der skal identificeres i forbindelse med skrivning af Fully Dressed Use Case, er interessenter, og hvad deres interesser er. Her er det ikke kun de åbenlyse, som firmaet og kunden, der skal identificeres, men også udenforstående interessenter, så som lovgivere, skat, banken osv.

I denne Use Case kan vi se, at lovgiveren har en interesse, da der er en lovgivning, der skal overholdes. Havde der nu været tale om et salg, så havde SKAT været en interessent.



Ligesom når vi skriver vores Vision, er det vigtigt at overveje sprogbruget i en Use Case. Der bør lægges vægt på, at ordene er så neutrale som muligt, og at ordvalget ikke låser sig fast på en specifik fremgangsmåde. Dette valg bør nemlig være op til designerne i en senere fase i forløbet.

Derfor har vi brugt ord som "anmoder" og "præsenterer" i stedet for fx "Sælgeren trykker på "Opret låneanmodning""



Use Cases bør desuden skrives uden diverse UI specifikke oplysninger. Endnu en grund til, at der står "anmoder om at oprette en ny låneanmodning" i stedet for "Trykker på opret ny låneanmodning", dette giver en langt bredere mulighed for nytænkning. Vi bør desuden anse systemet for en Black boks, og i væres beskrivelser være så uspecifikke som muligt.

Skrivning af Use Cases bør færdiggøres i Elaboration, og der bør kun tilstøde meget lidt arbejde m. Use Cases i Construction fasen.

### Brugergrænseflade (Louise)

Brugergrænsefladen er en essentiel del af programmet, og kan dog være noget vanskelig at udarbejde i en god format, der tilfredsstiller alle parter. Derfor er løbende brugertests en vigtig del af udviklingen. Fremstilling af modeller til dette, bør udvikles hurtigt, da man må påregne, at de forkastes om laves om af flere gange, efter samtaler med brugeren.

Det mest vellykkede er at have flere forskellige udkast, og ændre efter brugerens input, og herefter udfører testen igen.

Brugervenlighed er en meget vigtig faktor i vores system, og påvirker flere af de ikke funktionelle krav, både pålidelighed og ydeevne. Til gengæld modarbejder det sikkerhed, da vi ikke ønsker den øgede tilgængelighed.

Brugervenlighed øger succesgarantien i vores program, da manglende brugervenlighed vil øge brugerens modstand til programmet.

I vores tilfælde har vi valgt at lave mockups manuelt, for at speede processen op, sikre flest mulige inputs og give maksimal mulighed overblik og hurtigt at kunne ændre, efter utallige ændringer blev den endelige mockup rentegnet i Visio.

### GRASP (Louise)

GRASP, også kalder **"General Responsibility Assignment Software Patterns or Principles"** kan betragtes som et hjælpemiddel til at lærer "Responsibility-driven Design.

Når vi taler om Responsibility-driven Design (RDD),tænker vi på hvilke objekter, der har ansvar for hvad.

Der er 2 typer ansvar:

* Doing
* Knowing

GRASP er bestående af 9 grundlæggende mønstre. Der diskuteres endvidere, om der er tale om mønstre, eller om de reelt er gået hen og blevet principper, men dette er ikke vigtigt.

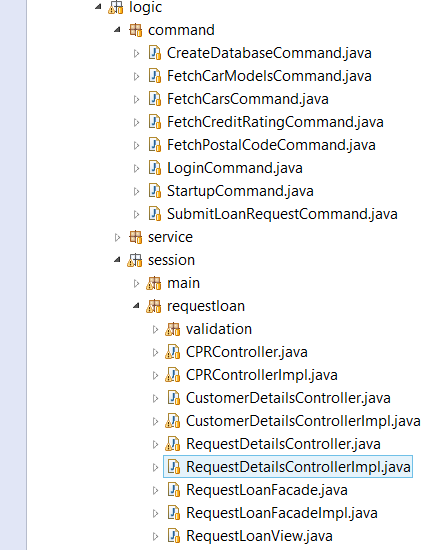
Når man taler om mønstre, er der tale om noget velkendt. Det er altså tale om noget der er gentaget igennem lang tid. Jo mere velkendt det er, jo bedre.

Et godt mønster har altså et sigende navn, en velkendt problemstilling, med en tilhørende løsning på dette problem.

De 9 GRASP mønstre er svære at sætte ind i individuelle bokse, da de langt hen ad vejen overlapper hinanden. Fx går "High Cohesion" hånd i hånd med "Low Coupling". Desuden kan "Low Cohesion" fx stamme fra en "Bloated Controller".

Når vi kigger i vores projekt, vil man meget hurtigt bemærke, at der er en overvældende mængde klasser. Dette er netop for at sikre, at hver klasse kun er ansvarlig for et konkret område. På denne måde, har vi sikret, at der er High Cohesion, og Low Coupling i programmet. Dette gør det både lettere at overskue og vedligeholde, da det gør den enkelte klasse meget lille.

Hvis vi forestiller os, at de Controllere, der ligger i session.logic



Hvis vi forestiller os, at vi i stedet for disse Controllere, havde lavet én stor Controller, der skulle tage sig af delegering af alt i FFS-01, så havde vi haft en klasse, der havde indeholdt mange hundrede SLOC. Dette er hvad vi ville kalde en "Bloated Controller", klassen ville på denne måde have alt for mange ansvarsområder, og komme til at lide under både Low Cohesion og High Coupling.

En anden måde at gøre vores Controllere til en Bloated Controller, ville være at lade Controllerne udfører opgaverne selv, i stedet for at delegere. På denne måde ville klassen igen have et alt for stort ansvarsområde, og herved igen lide under Low Cohesion.

### Information Experts (Louise)

Sammenhørighed er ikke noget, der kan vurderes alene i et program, man vil som tidligere set logisk kigge på koblingen, men eksperter er endnu en del, som vi overvejer sammen med. Disse kalder vi "Information Experts"

Pointen m. Information Experts er, logisk at kigge på, hvilken klasse, der bør være ansvarlig for hvilken information.

*Vi kan fx kigge lidt på LoanRequest.*



Rent principielt, kunne vi godt komme al information, der kommer til at berører LoanRequest-klassen ind i LoanRequest, men dette vil skabe et par problemer, som vi allerede har snakket om:

* Low Cohesion
* High Coupling

Dette ville fører til en meget stor og uoverskuelig klasse, der desuden ville være svær at vedligeholde, og blive påvirket ofte, hvis der skal laves om i programmet.

Dette leder os til Information Experts.

I vores eksempel, kan vi se, at LoanRequest har kun ansvar for at kende til en dato, en ønske afdragsperiode og en ønsket tilbagebetalings sum. De resterende informationer er mere logiske at lægge i et salg. Heraf vores salgs-klasse.

I den er der igen kun få ting, der behøver være kendt i salg. Det samme gør sig gældende for de resterende 4 klasser.

Der er endnu flere klasser, der indeholder små mængder information, der logisk set bør ligge ved dem.

Dette hjælper til, at det er let, ud fra klassens navn, at identificere, hvad der vil ligge i den pågældende klasse.

### Whitebox test (Louise)

I modsætning til Blackbox test, er Whitebox test en test, der er baseret på det fysiske system, og ud fra dette udarbejdes der testdata.

Det er normalt programmører og teknikkere, der udfører whiteboxtest og komponent test.

Der bliver talt om 3 typer test:

* Statement Coverage - Hvor det gælder om, at sætte sig et mål (typisk er målet 100 %) om hvor stor en del af koden, man ønsker at afdække.
* Decision Coverrage - Også kaldet "Branc Coverrage". Dækker forgreninger i systemet. Der er ikke tale om almindelige IF sætninger (målet her bør også være 100 %)
* Path Coverrage - Gælder om at afdække alle mulige veje til det samme mål. Her kan det være sværtat opnå 100 %, men dette bør stadig være målet.

Vi taler desuden også om Condition Coverrage, der afdækker tests af programmet i forskellige stadier.

### Reviews (Louise)

En af de ting, som vi med fordel kunne have fokuseret kraftigere på, er reviews.

Reviews er vigtige til udførsel af tidlig test.

Når man udfører reviews, sætter en række mennesker sig sammen i en gruppe og gennemgår sammen produktet.

Dette er dog lykkedes os at gennemfører en form for reviews af mange af vores modeller, i form af samlet gennemgang af de enkelte modeller. Derfor har vi fået en del af bonus af både erfaringsudveksling, etablering af standarder og bedre og mere ensartet planlægning.

Vi kunne dog formentligt have haft megen gavn af den strukturerede review, indlagt allerede fra start.

Hvis man har et ønske om, at udfører tidlige tests kan der desuden bruges:

* Prototyper
* Tænk højt test

Et sted hvor reviews desuden giver god mening er i forståelse af forskellige termer. Det kan være svært på egen hånd, at identificerer de ord, der ikke er enighed omkring. Men når en review-gruppe sidder samlet, vil det hurtigt vise sig, hvilke ord der falder udenfor.

Reviews hjælper til:

* 0 - fejls udvikling
* Opfyldelse af krav
* Opretholdelse af standarder
* Ensartethed

Når vi taler om reviews, så taler vi endvidere også om Verificering og Validering, der ikke bør forveksles.

Definitioner:

* Verificering - Når udvikleren gennemgår data- og klasser modeller, for at sikre, at det er dokumenteret på den korrekte måde.
* Validering - Når brugeren gennemgår systemet, fx vha. prototyper, for at sikre, at det er de korrekte handlinger der udføres, vha. virkelighedstro scenarier.

Ud fra disse to definitioner, kan vi konkludere, at vi har verificeret dele af vores system, men at der er en overvældende mangel på validering.

Der findes flere typer test, der kunne være udført

En af de typer er Walk-Throught

formålet her er at se, om programmet fungerer i virkelige scenarier. I vores ville dette være at lege sælger, og derefter gennemfører fx en oprettelse af en låneanmodning. Herved kunne vi se om programmet er i stand til at håndterer dette.

Vi er her nødt til at have brugt nogen tid på forberedelse og identificere, hvilke fejl, der kunne opstå.

Denne type test er procedureorienteret. Dvs. at det skal udføres i den rækkefølge programmet skal afvikles.

Denne type reviews kan udføres på alle dynamiske modeller. Dvs. sekvensdiagram, prototyper el. use cases.

Man kan også udfører Play-Throught, som på mange måder minder om Walk-Throught, med den forskel, at det i denne type er mennesker der "spiller" de forskellige roller i systemet.

Vi lavede den fatale fejl i vores projekt, at vi ikke fik skrevet reviews på projektplanen som en selvstændig opgave, ergo havde vi ikke den nødvendige tid til udførslen af dem.

### Construction (konstruktionen) (Louise)

Construction er en meget stor fase, dette behøver dog ikke afspejle at der ligger kalendermæssigt meget tid i den. Når man træder ind i denne fase er alt beskrevet, og der skal blot kodes (og selvfølgelig holde alle beskrivelser up to date).

I denne fase kommer der ofte mange ekstra mennesker på projektet. Da det her både er muligt at sætte mindre erfarne programmører på eller benytte sig af outsourcing.

I denne fase ligger al det sidste kode, samt en masse test. Det gælder om hurtigst muligt at få et funktionelt program til verden, så der kan påbegyndes test af dette.

Når man forlader denne fase, skal der tages stilling til, om systemet er klar til en release. og om kunden er klar til at få programmet i Transition

I et så lille projekt som vores, er construction ikke en fase, man behøver at gå i. Da vi meget sent i forløbet har alt fuldt beskrevet.

### Transition (overdragelsen)(Louise)

Transition er den sidste fase. Formålet med denne fase er, at få produktet ud til brugeren. Når dette sker vil der oftest tilstøde problemer el. lign, der skal omkodes, laves ny release el. færdiggøres nogle features.

I denne fase vil der ofte ligge en el. anden form for beta test, så man ved, at programmet stemmer overens med kundens forventninger. Desuden vil man sørge for oplæring af fremtidige brugere, og at få kørt programmet ud til alle fremtidige brugere.

I nogle projekter vil dette bare anmærke starten på et nyt projekt, Da der blot startes på udvikling af næste version.

I vores projekt har vi ikke haft en Transition fase, og det tætteste vi kommer på, må siges enten at være aflevering af projektet el. selve eksamen.

1. http://da.wikipedia.org/wiki/Unified\_Process#/media/File:Development-iterative.png [↑](#footnote-ref-1)