# Hvad er IT:

Hvad er IT(informations teknologi):

Input/output, enhed der behandler data og information, hukommelse. Netværk/ kommunikation.

* *Brugen af computere til at gemme, udlede, kommunikere og manipulere data og information.*

ICT(c = kommunikation)

**IT system(computerised system):** Et software system i stand til at lave IT.

* *Et organiseret system til at gemme, udlede, kommunikere og manipulere data og information.*

**Andre mulige elementer (Ikke definitioner brugt i kurset):**

* Eksterne enheder
* Andre systemer, hvad kommunikerer systemet med
* Brugere og andre personer, hvem bruger systemet
* Procedure, hvordan bruges systemet
* Reglerfor hvordan man bruger systemet

**App-domain:**

Hvor brugere er. Organisationen der administrerer, overvåger eller kontrollerer et problemdomæne

**Problem domain:**

Delen af konteksten der er administreret, overvåget eller kontrolleret af et system. Når noget sker i problemdomænet, opdateres modellen og app domænet.

Problem domæne 🡪 Model 🡪 App-domæne

Opdateringen fra problemdomænet til modellen, og fra modellen til app domænet kan variere i vigtighed af hastigheden. Det er noget man bestemmer når man laver requirements.

Det sammen objekt kan være en del af begge domæner, men ikke altid. Det handler om hvad menneskers rolle er i handlingen.

Man kan have en model med en funktion der automatisk opdaterer modellen, der igen opdaterer problemdomænet.

**Eksempel:**

Hvis en ansat på et universitet tager et kursus, vil denne både være en del af problem og app domænet – både en del af systemet og administreret af systemet.

I problemdomænet vælger en underviser en bestemt karakter til brugeren. Nu opdateres modellen, og fortæller karakteren. Denne vises i app domæne. Hun går altså ikke gennem problemdomænet og beder om karakteren, men får den gennem modellen. Her er hastigheden af opdateringen ikke kritisk. Det kan den fx være i et system for at tilkalde ambulancer.

Lærer informerer om karakter 🡪 Systemet ændrer karakteren 🡪 Brugeren ser karakter

**Bruger:**

Er i app domæne, og får info om problem domænet gennem modellen.

Fire perspektiver:

**Informations perspektiv** - skal tilbyde en brugbar model til problemdomænet. Er vigtig i analysen, men også med design da dette skal gøre modellen mere effektiv og brugbar.

**Brugerens synspunkt:** systemet skal integreres i app domænet. Forstå de personer og andre systemer der skal benytte systemet. Skal kunne samarbejde med andre enheder og ændre sig i forhold til ændrede krav.

**Arkitekturelle perspektiv:** systemet skal køre på en bestemt platform. Hvordan skal systemet opbygges osv. Husk de fysiske komponenter i dette.

**Et hele:** det skal virke.

Emphasize the architecture:

Fokuser på systemet fra starten. Brugeren og andre systemer ser kun interfaces, under det ligger alle funktionerne, og som fundament af det, modellen.

Arkitekturen er en generel struktur af det der bliver udviklet videre

Systemet er den samling af komponenter der implementerer

**Fire hovedaktiviteter:**

**Komponent design:**

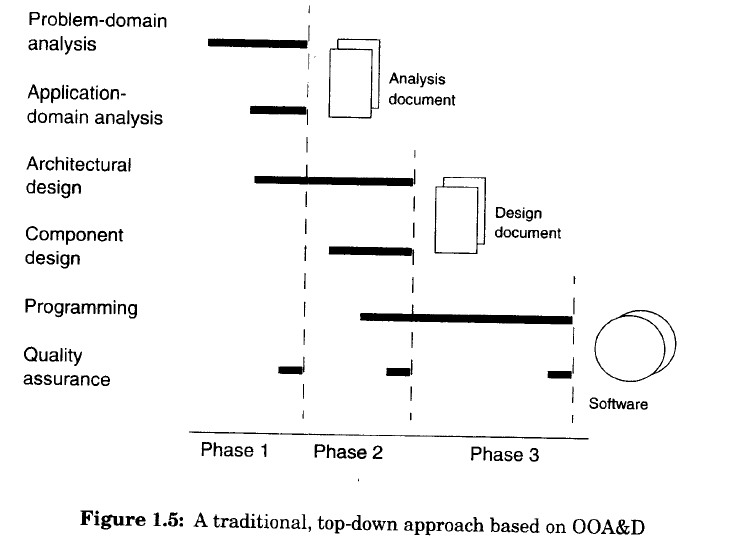
**Applications domæne analyse:**

**Problem domæne analyse:**

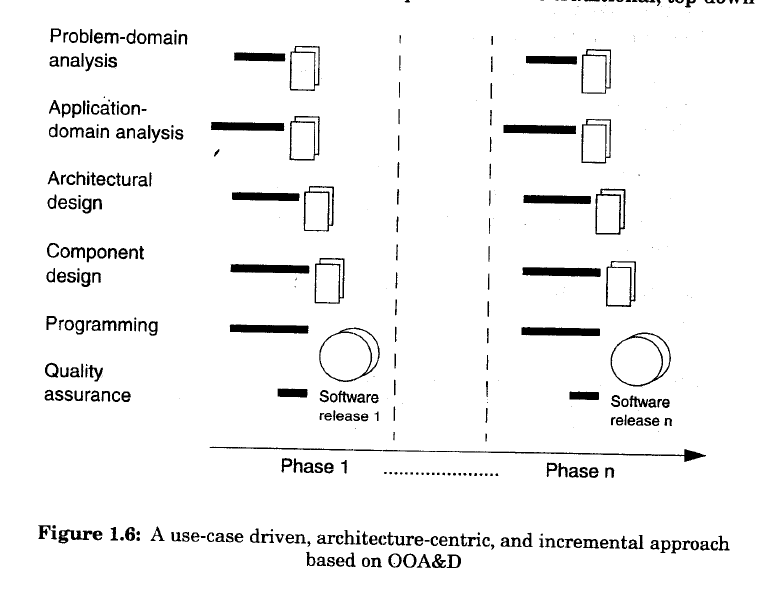
**Komponent design:**

**Fremgangsmåder:**

**Traditionel top-down:**



**UML**



Top down er mere drevet af problem domæne analyse, mens den anden er drevet af application domæne analyse

**Dokumentation:** giver kontinuitet i løbet af systemudvikling.

**Exhaustive seach = brute force,** buttom up problem solving

**Metode:** Retningslinjer for at udføre udviklings aktiviteter(fx analyse og/eller design).

# System valg:

Formål: Være enig I de overordnede karakteristika af systemet.

Konsept: definition af systemet. En konkret beskrivelse af et computersystem i et normalt sprog.

Principper: forstå situationen. Udvikle nye idéer. Definere alternative systemer.

Resultat: en system definition der udfylder FACTOR criterier.

Når man udvikler et system skal man overveje følgende:

* Hvilke problemer er det vi vil løse
* Er det planlagte system en brugbar løsning
* Hvad ville der ske hvis vi udviklede et helt andet system?

## System definition:

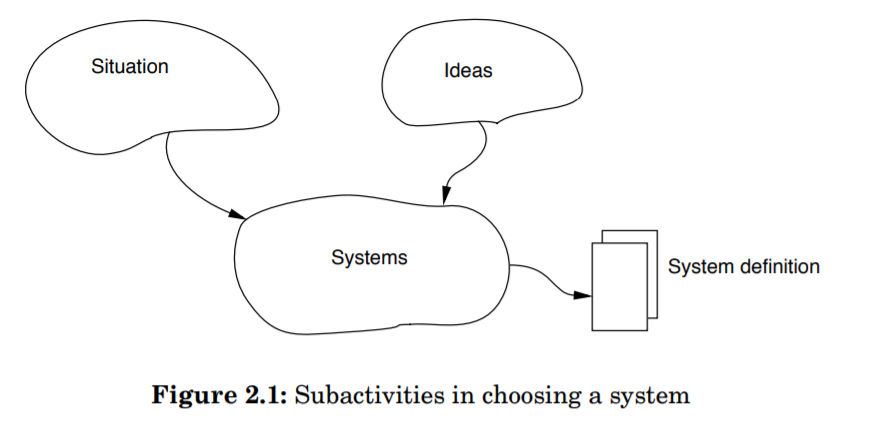
Nogle gange er problemet systemet udvikles til præcist defineret, mens det andre gange er mere løst. Et problem kan altid blive set på forskellige måder, enten fordi det er af forskellige personer, eller forskellige vinkler. SD udtrykker de fundamentale egenskaber af systemudvikling og brug. Det viser systemet i kontekst, med hvilken information de burde indeholde, hvilke funktioner det skulle tilbyde, hvor det skal bruges og hvilke udviklings betingelser der gælder. Idéen er at man finder forskellige måder og muligheder at løse problemet på. Du kan se forskellige løsninger, og sammenligne alternativer. System definitionen der bliver valgt skulle lægge et godt fundament for den videre analyse og design aktiviteter. Skal være kort of præcis, og indeholde de fundamentale valg omkring systemet. Det giver overblik, og gør det let at sammenligne.

Hvis kriterier skal genovervejes, brug fx en iterativ fremgangsmåde, hvis ikke brug vandfald.

Uklare kriterier er blevet antaget som hovedkilden til problemer i systemudviklingen.

Når man skal overveje om ting i app og problem domænet er relevante, kigger man på SD

## Valg af system:



De tre underkategorier når man skal vælge et system er:

1. Situationen. Prøv at danne overblik over situationen og de forskellige måder at tolke denne.
2. Idéer. Teknikker til at understøtte kreativitet og introducere nye måder at tænke på.
3. Definition. Diskussion og evaluering af alternative definitioner af vores situation

En definition vælges ved at gå igennem de tre underkategorier. Når man beskriver en situation giver det ofte muligheder for nye idéer. Denne proces påvirker de andre underkategorier, og giver omstændigheder der skal undersøges, og idéer der skal yderligere undersøges.

Valg af system kræver: fantasi, gode idéer, udviklingsmæssige evner og kritisk sans.

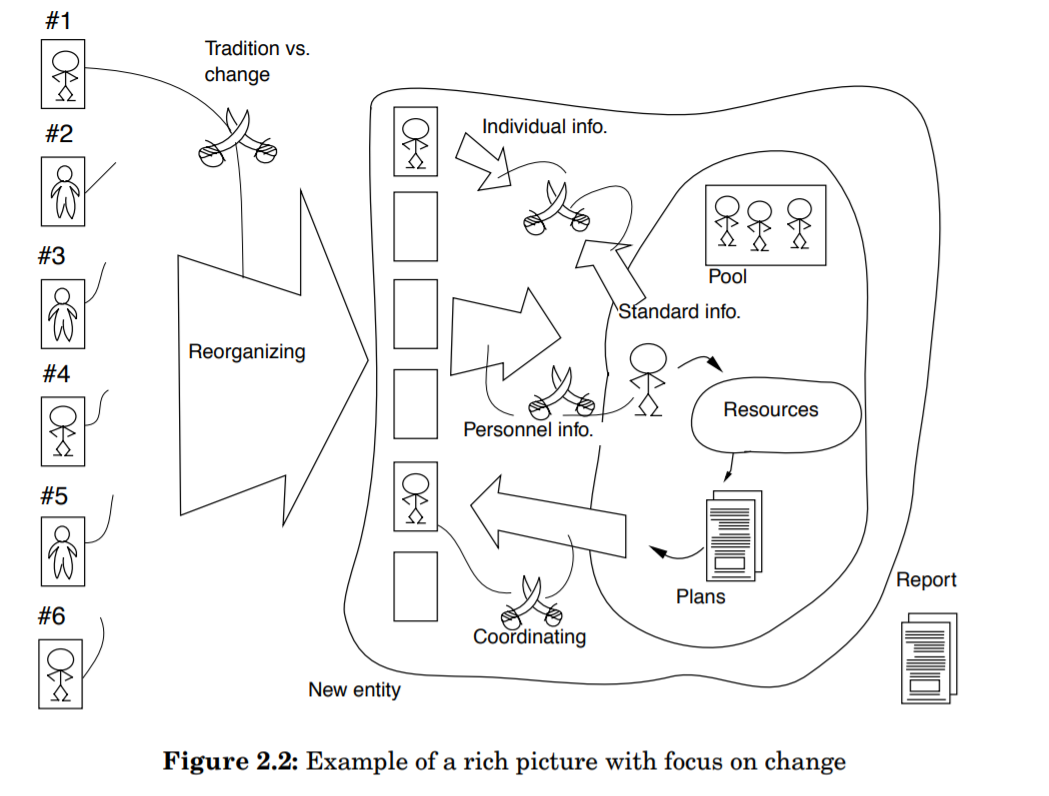
Man skal starte med at undersøge brugerens situation, og forstå hvori udfordringen ligger. Dette kræver tæt samarbejde med brugeren og kunden. Man skal reflektere kritisk over brugerens ønsker og idéer, især hvis de peger mod en bestemt løsning. Brugeren har ofte ikke tilstrækkelig forståelse af udviklingsprocessen.

## Beskriv situationen:

Man skal forstå brugerens situation ordentligt. For at opnå dette skal man være klar til en konstruktiv diskussion.

### ’**Rich** Pictures’:

Et ”rich picture” er en beskrivende tegning der repræsenterer illustrators forståelse af en situation.

Eksempel på et rp fra et hospitals udvikling af et projekt(side 27): 

Billedet beskriver en ændring i seks autonome afdelinger til en ny struktur hvor de bliver lagt sammen. Formålet er at øge koordination og samarbejde mellem afdelingerne.

Rp fokuserer på vigtige aspekter af situationen. Det skal give en bred beskrivelse af situationen, så man kan have flere forskellige fortolkninger.

Det kan være praktisk at differentiere mellem rp der fokuserer på stabilitet og rp der fokuserer på ændringer. Det er vigtigere at forstå de essentielle aspekter end at arbejde systematisk. For at forstå kulturen og finde ud af hvad der er vigtigt skal man besøge organisationen og se hvordan den virker, tale med forskellige mennesker om hvad der sker og burde ske, og måske lave nogle interviews. RP kan være brugbar til at lave god kommunikation bed brugeren. De kan også bruges til at beskrive syn på situationen mellem udviklere.

### At tegne rich pictures:

Det er en god ide først at tegne vigtige ting så som mennesker, fysiske objekter, steder, organisationer, roller og opgaver. Problem ejere kan også være en del af billedet. De er dem er identificerer problemet, direkte eller indirekte. Kunden er en problem eje, men det er brugeren også måske. Man skal ikke tegne et rp som kunden udtrykker problemet, men som man selv opfatter situationen. Det er vigtigt at finde gode symboler på elementerne.

### Processer:

Når objekterne er tegnet skal relationen imellem disse udarbejdes. Processer er de mest fundementale relationer mellem elementer. En proces beskriver aspekter af situationen der ændrer sig, er ustabile eller under udvikling. Disse kan illustreres med pile. Når man beskriver udveksling af information er det vigtigt at forstå hvordan denne information er produceret, forstået og brugt.

### Strukturer:

Beskriver aspekter af en situation der opfattes som stabile eller svære at ændre. De beskrives enten ved at tegne steger mellem elementer, eller ved at placere relaterede elementer i en fælles figur som firkant eller cirkel.

### Problemer:

Typisk værdier og statements der karakteriserer et problem. Kan repræsenteres som krydsede svær. Problemer kan markerer utilfredshed, i stedet for deciderede problemer. Det er derfor en god idé at knytte et par ord til et problem. Undgå at beskrive løsninger samtidig. Fokus skal være på hvordan bruger ser problemet.

### Praktiske råd:

Et brugbart RP skal:

* Have en masse information og være åben for fortolkning.
* Præsentere proces of struktur i en sammenhængende og velbalanceret måde.
* Vise mindst et problem
* Pege på flere relevante computer systemer.
* Være rigt, ikke kaotisk.
* Belyse nøgle aspekter af situationen så den kan forstås på mange niveauer
* Ikke repræsentere dato og data processer

Det er bedre at tegne flere billeder end at gøre det kaotisk, især hvis situationen er meget kompleks.

# Få idéer:

Vær åben for nye idéer. Gamle idéer er god inspiration. Balancer mellem at udvikle nye idéer og genbruge gamle. Nye idéer udvikles i samarbejde med fremtidige brugere. Uden forståelse for brugerens kontekst vil idéer ofte være generelle og abstrakte.

Det er foreslået at bruge

* Eksempler
* Metaforer
* Eksperimenter

Når man udvikler idéer sammen med brugere.

### Eksempler

Det er en god ide at kigge på systemer der minder om dem brugeren ønsker.

For hver eksisterende løsning besvares følgende spørgsmål:

* Hvilke ideer er fundament for systemet
* Virker ideerne brugbare? hvorfor?
* Vil ideerne virke i din kontekst? Hvorfor?
* Kan ideerne blive ført hen på dit system? Hvordan?

### Metaforer

Her ses systemer i et nyt lys. Dette er kun en intellektuel øvelse, der ikke involverer andre. Kan hjælpe med at overføre ideer fra andre områder. Når en metafor er fundet, følges følgende step:

1. Lav en lang liste af aspekter relateret til metaforen
2. Overfør aspekter til dit system
3. Bestem hvilke aspekter der er brugbare

### Eksperimenter

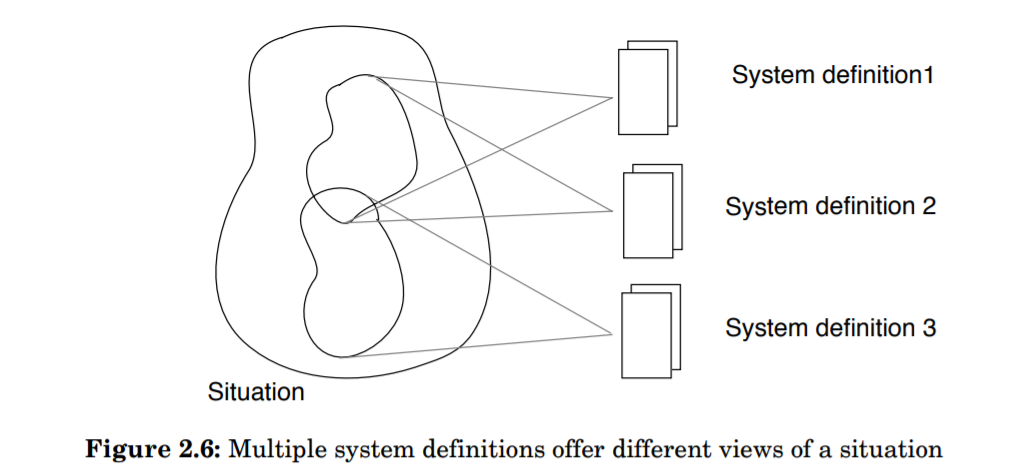
Man kan bruge en simpel prototype til at forsøge sig frem. Dette kan også være med en tegning for at undersøge display. Dog er det bedre hvis muligt at lave computerbaserede prototyper.

# Definer systemer:

Her vælges systemet. Dette gøres ved systematisk at redegøre for fortolkninger, muligheder og konsekvenser af forskellige løsninger. Man kan formulere kvaliteter af potentieller systemer. Denne kan gøres ved at formulere kvaliteterne til systemdefinitioner, der er nemt overskuelige

## Systemer er holistiske syn:

Det samme system kan blive set på forskellige måder alt efter hvordan det er brugt. Hver definition repræsenterer en opfattelse af verden filteret gennem ideer, uddannelse og baggrund for de involverede.



En systemdefinition skal udtrykke systemfokuserede kvaliteter, og ikke detaljerede komponent baserede kvaliteter. Før man vælger et system skal man arbejde med flere forskellige.

## Eksempler på definitioner, side 38.

## Praktiske råd:

* Brug generelle termer, og sæt fokus på vigtige definitioner
* Fokus på ideer og ikke beskriv situationen som den er
* Lav definitionen kort og præcis
* Eksperimenter med forskellige definitioner
* Brug nye måder at tænke på
* Brug udvælgelses processen til at lægge mærke til nye relevante aspekter

# Factor standard:

FACTOR standarden består af seks elementer.

1. Funktionalitet. Systemfunktionerne skal understøtte hovedopgaverne af applikationen
2. Applikations domæne. De dele af organisationen der administrerer, overvåger og kontrollerer problem domænet
3. (Condition)Betingelser. Betingelserne hvorunder systemet skal udvikles.
4. Teknologi. Både den til at udvikle systemer, og den hvorpå systemet vil køre.
5. Objekter(Problem domæne). Hovedobjekterne i problem domænet
6. (Responsibility)Ansvar. Systemets generelle ansvar i forhold til dets kontekst.

FACTOR kan bruges til at understøtte system definition udviklingen, hvor man omhyggeligt undersøger hvert element. Det kan også bruges til at starte beskrivelsen af systemet og bruge kriterier til at se hvordan systemdefinitionen understøtter de seks faktorer. Begge begyndelsessteder kan bruges. Det kan godt være at man flere gange skal formulere definitionen, og derefter sammenholde den med FACTOR.

Hvad gør systemet, og hvad gør brugeren? Definer grænsen mellem disse.

## Evaluering og valg:

Systemudvikleren skal ikke vælge et system, men at lette valget. Et system skal vælges gennem aktive forhandlinger imellem alle involverede parter.

# Principper:

## Genbrug af mønstre:

Ikke lav en ansat og elev klasse, en person kan være i begge roller. Lav i stedet en klasse med et antal elev roller, og en række ansat roller. Find ud af om objekter kan have forskellige roller. Sørg for at alle kan være i et objekt.

# Prototype

## Forskelle:

Hvor et system har interface, funktion og en model, har prototype måske kun en eller to af disse, som fx interfaces. En prototype kan være teknisk begrænset, og mangle fx en sluknings funktion.

Det hedder: Interface prototype og tekniske prototype.

Brugeren kan give feedback på designet før udviklingen. Brugere ved sjældent præcis hvad de vil have.

## Typer af eksperimenter!!!:

### Quick and dirty:

Implementation uden analyse og design Genudviklet indtil brugere er tilfredse. Kan blive meget dyrt

### Throw away

* UI protoryper
* Mock-up

### Design – drevet:

Teknisk

### Mock-up:

Non executional model, kun til visuelt design.

### Evolutionary:

Lave kernen af systemet, og udvikler det stille og roligt. Bliver stille og roligt den sidste version.

## Explanatory:

Bruger prototype til at udvikle nye design idéer.

Formålet er altid at lære noget.

De steps man skal følge er:

1. planlægning

* Hvad er fokus af eksperimentet
* Hvad er blevet udelukket
* Hvilke forudsætninger er der

1. Udvikling

* Man skal lave den hurtigt

1. Forberedelse

Man skal bestemme sig for følgende ting:

Samarbejde, Man kan fx lade brugere teste prototype under observation

* Realisme (Hvad sker der i den virkelige verden), man kan enten lade brugere bruge prototype i forbindelse med den normale aktivitet, eller separeret fra denne.
* Udvælgelse, hvem skal teste. Hvem repræsenterer de

Er der noget bruger skal forberede

1. Test

Realiser de ting valgt før. Dokumentation kan fx ske ved hjælp af en dagbog brugeren skriver,

1. opsummering

Opsummer på det valgt i planlægning. Her kan man enten vælge at lave flere tests, eller begynde at udvikle videre.

## Evaluating:

Bruger to eller flere prototyper til at vælge design.

## Hvad kan smides væk:

Udvikling

Læg det på en anden platform i forbindelse med prototype hvis lettere. Man skal ikke lave så meget kvalitets sikring og dokumentation(undtagen evolutions)

Produkt:

Skal ikke være så meget kode.

Behøver ikke at være effektiv, pålidelig og rubust hvis det ikke er en teknisk prototype

Ikke vigtigt med errorhandling.

Horisontal slice, vise fx kun ui, ingen funktioner, flere inteface sider uden funktionelt indhold.

Vertikal slice, Nogle funktioner og lidt model, lidt interface. Fx kun en side i en app.

## Implementation af prototyper:

Papir: godt for visuel, dårligt for funktion. Wizard of Oz hvis illusterer struktur. Godt til at vise design

Power point: lidt hurtigere visuel tilgang.

Tools: marvelapp.com, justinmind.com, Balsamiq(koster penge)

Jo tidligere i processen du er, jo længere er du oppe på listen.

# Boehm’s specific technique/ constructive cost model/ cocomo

Empirisk omkostnings estimat model. Bruger data fra forrige input til at beregne næste. Det endelige estimat bliver dannet ved at korrigerer det første.

## Wizard of Oz prototype:

Få det til at se ud som om noget er funktionelt uden at der virkelig er noget IT teknologi bag

# Iterativ fremgangsmåde:

Grund: Faktiske problemer er sjældent klare og præcise. Hvis man prøver noget gentagende gange, reducerer man usikkerheden. Når man siger at prototypen er god nok kan man begynde at arbejde med det. Mange gange laver man en release, får feedback og korrigerer systemet – software opdateringer. Prototyper er en del af dokumentationen for krav. Her kan man sige ”i godkendte det”.

## Contingency teori:

Hvordan vælger man mellem vandfald og ”life cycle”(iterativ) prototype.

Handler om usikkerhed.

* Organiserings og teknisk kontekst af systemet
* Fremtidig computer system
* Erfaring og evner af bruger
* Erfaring og evner af udvikler

Baseret på usikkerhed. Hvis lav usikkerhed, så kan kriterie bestemmelse baseres på en informeret fremgang på analyse og eksisterende systemer

### Davis

Hvis høj, brug specifikationer og prototyper.

### Burns og Dennis:

Handler om kompleksitet mellem kompleksitet og usikkerhed

Usikkerhed:

* Hvor struktureret er brugers arbejde
* Graden af forståelse brugeren har af deres arbejde
* Hvad ved vi og hvor meget erfaring har vi på området?

(lavere, mere usikkerhed)

Kompleksitet:

- projekt størrelse

- antal af brugere

-volumen af ny info

- kompleksitet af ny info

### Mathiassen og Stage:

Det skal gøres ofte, spørg ofte hvad usikkerhed og kompleksitet.

Hvis høj usikkerhed, prøv igen og igen.

Når man laver løsninger for at reducere kompleksitet, kommer nye kilder for usikkerhed og omvendt. Fx, jo flere man spørger, jo mere kompleksitet og mindre usikkerhed

At dele projekt ind i elementer giver mindre kompleksitet og mere usikkerhed. Man skal ende med nul usikkerhed, og nul kompleksitet

Skal bruge prototype ved høj usikkerhed. Giver mere kompleksitet.

Ved høj usikkerhed:

Ved høj kompleksitet:

Risiko analyse faktorer:

Definer rissisi.

Specifiser konsekvenser

Vælg løsnings stategi.