# Computationel tænkning

Udarbejdet af Lars Skjærbæk, UCH

Computationel tænkning er nogle teknikker som gør det nemmere for os at omsætte et behov eller en ide til en IT-løsning. Metoden omfatter 4 hovedteknikker:



**Dekomposition** er nedbrydning af en opgave i delopgaver. Ved at nedbryde opgaven i små overskuelige delopgaver, kan vi på en struktureret måde komme rundt omkring alle elementerne i løsningen, og få diskuteret alle detaljer, uden at miste overblikket. Til dette kan man benytte teknikker såsom struktur-diagrammer og flowdiagrammer. Til de lidt mere avancerede opgaver kan man anvende USE-cases, data flow diagrammer eller objekt orienteret analyse.

**Abstraktion** **og generalisering** handler om at fokusere på de væsentlige elementer. Hvis man skal vide alt om binære talsystemer, internet protokoller og programmering for at lave en hjemmeside, så kommer man aldrig i gang. Heldigvis er der lavet generelle løsninger, der gør, at udviklerne kan abstrahere fra dette, så de kun skal koncentrere sig om designet. Gennem abstrakt tænkning kan vi fjerne alt det unødvendige (eller bare det vi ikke behøver at løse lige her og nu) og så bør vi ellers forsøge at lave løsningerne generelle, så samme løsning kan bruges flere steder. I forbindelse med programmering laver vi abstraktion gennem brug af funktioner.

**Mønster genkendelse** er når ting går igen. Er der noget computeren er god til, så er det at lave gentagelser (løkker). Derfor skal vi analysere problematikken og kigge efter gentagelser, for derefter at skrive koden ved brug af gentagelser.

**En algoritme** er en opskrift på hvorledes man løser et konkret problem. Det kan eksempelvis dokumenteres i et flowchart, hvor løsningen beskrives trin for trin. Det kan også dokumenteres ved brug af formler og regler. (bemærk at der i små opgaver er en del sammenfald mellem dekomposition og algoritme design).

De fire teknikker er nogle metoder, vi kan bruge, når vi skal angribe et problem (gælder ikke kun IT systemer). Der er ikke nogen bestemt rækkefølge de skal anvendes i, men ofte er det en god ide at starte med dekomposition. I nedenstående eksempler viser jeg hvorledes computationel tænkning er anvendt i forbindelse med udvikling af hhv. en webshop og en app

# Eksempel 1: Udvikling af webshop

**Dekomposition** er nedbrydning i de sider, som skal være på ens website. Til dette kan vi lave et strukturdiagram. Der skal også laves en proces for, hvorledes selve købsprocessen skal gennemføres. Dette kan dokumenteres i et flowdiagram.

Produkterne skal organiseres, men bortset fra det vil alle produkter formentlig skulle præsenteres på samme måde (**mønster genkendelse**). Desuden er der allerede lavet ”standard” versioner af webshops, som kan benyttes, med mere eller mindre standardiseret design og funktionalitet. Vælger man en standard løsning kommer man hurtigt i gang.

En betalingsløsning er ret kompliceret at udvikle, men heldigvis kan vi købe dette som en service. Derved kan vi **abstrahere** fra problematikkerne omkring dette. Benytter vi standard løsninger kan vi helt abstrahere helt fra at skulle kode.

I forbindelse med indkøbskurven skal der laves **algoritmer** til at beregne fragt, moms og det samlede betalingsbeløb. Måske skal der også laves en valuta omregning. Også dette er standard i en webshop.

En prototype på en E-shop kan udvikles på WIX.COM. Der er masser af tutorials på Youtube, som viser, hvorledes man kan lave en webshop i WIX. Vælg et design med en shop, og gå så i gang med at lægge produkter ind. Der kan arbejdes med optimering af billeder til web, samt optimering af tekster, så man får bedst mulig placering på søgemaskinerne. Vil man idriftsætte shoppen, må man mod betaling tilknytte et domæne og en betalingsløsning. En sådan løsning er nem at starte ud med, men besværlig når vi skal til at ekspedere varerne, da løsningen ikke er bundet sammen med økonomi og logistik.

# Eksempel 2: Udvikling af kasseapp

**Dekomposition:** Først nedbryder vi opgaven i delopgaver ved at lave et flowchart, som viser processen. (Dette kan gøres i plenum sammen med eleverne. Bed dem evt. tegne et flowchart på papir eller i webapplikationen [draw.io](https://www.draw.io/)) Resultatet er vist i nedenstående flowchart.



”Scan vare” kan foregå på mange måder. Det kan være noget med at læse en QR-kode, og så slå prisen op. Det kan også bare være et simpelt input felt, hvor vi indtaster prisen. Vi vælger at **abstrahere** fra de mange muligheder, og laver i første omgang den enklest mulige løsning, nemlig et input felt, hvor prisen tastes ind.

Beregn sum er en meget simpel **algoritme** (sum = sum + pris). Vil man trække noget fra igen, må man indtaste en negativ pris.

”Betal med kort” er en meget avanceret proces, som der heldigvis er lavet standardløsninger for. Vi vælger i dette eksempel at **abstrahere** fra dette. Betal kontant er en manuel proces. Vi kan evt. senere udvide app’en til at beregne byttepenge.

Du kan under kode delen se, hvorledes kasse app’en er implementeret.

Som alternativer til kasseapp’en kunne man også lave apps til valutaomregning, momsberegning, låneberegninger, tidsregistrering, kørselsregistrering, rejseafregning m.m.