DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Konstruksjoner og materialer/ Byggkonstruksjoner	Vårsemesteret, 2022 Åpen
Forfatter: Sondre Rosten Ohma	Sondre Rosten Ohma (signatur forfatter)

Fagansvarlig: Sudath C. Siriwardane

Veiledere: Ashish Aeran, UiS

Daniel May Instanes, Norconsult AS

Tittel på masteroppgaven:
Forslag om bruk av Dynamo for en effektiv arbeidsflyt mellom Revit og FEM-Design

Engelsk tittel:
Proposal to use Dynamo for an efficient workflow between Revit and FEM-Design

Studiepoeng: 30

Emneord: Dynamo, Revit, FEM-Design, arbeidsflyt, Parametrisk design, effektivisering, Optimalisering, armeringsmodell, StruXML	Sidetall: 80 + vedlegg/annet: 20 Stavanger, 15.06.2022

Forord

Denne masteroppgaven er utarbeidet som avsluttende oppgave for min mastergrad i konstruksjoner og materialer ved Universitetet i Stavanger, fordypning Byggkonstruksjoner.

Byggebransjen er under en enorm teknologisk utvikling som åpner opp for nye metoder som kan bidra til en mer effektiv prosjektering. Gjennom mine fem år med studier er dette et tema jeg har hatt stor interresse for. Dermed ønsket jeg å benytte masteroppgaven til å utforske noe innenfor dette temaet, som samtidig er fremtidsrettet. Her er parametrisk design et fremadstormende verktøy som benyttes for å effektivere prosjekter, og noe som jeg ønsket å fordype meg i.

Da jeg hadde sommerjobb som rådgivende ingeniør, utførte jeg beregninger i FEM-Design for å finne nødvendig armering. Deretter skulle jeg manuelt presentere armeringen i en 3D-modell i Revit. Jeg ble nysgjerrig på om det var mulig å etablere et Dynamoscript som kunne automatisere denne prosessen. Dermed ble dette inspirasjonen bak oppgaven min.

Oppgaven er utarbeidet i samarbeid med Norconsult AS med hensikt om å utforske hvordan Dynamo kan etablere en arbeidsflyt mellom Revit og FEM-Design som bidrar til en en mer effektiv prosjektering.

Jeg ønsker å takke intern veileder ved Universitetet i Stavanger, Ashish Aeran, for god oppfølging og tilbakemeldinger gjennom oppgaven. Jeg vil også rette en takk til ekstern veileder ved Norconsult AS, Daniel May Instanes, for god hjelp i programvarene som benyttes i oppgaven. Daniel sin faglige kompetanse har vært til stor hjelp for å finne gode løsninger gjennom oppgaven.

Stavanger, 15. juni 2022

Sondre Rosten Ohma

Sondre Rosten Ohma

Sammendrag

Byggebransjen er under en enorm teknologisk utvikling som åpner opp for nye metoder som bidrar til en mer effektiv prosjektering. Her er parametrisk design et fremadstormende verktøy som benyttes til dette formålet.

Denne masteroppgaven tar for seg hvordan parametrisk design kan benyttes for å bidra til en mer effektiv prosjektering. Det skal brukes et visuellt programmeringsverktøy, Dynamo, for å etablere en arbeidsflyt mellom tegningsprogrammet Revit og beregningsprogrammet FEM-Design. Her skal det settes søkelys på overføring av elementer mellom programmene, der armeringsprosessen er sentral.

Med bakgrunn i dette er det utarbeidet to script i oppgaven. Det første scriptet tar for seg overføring av en analytisk modell fra Revit til FEM-Design. Scriptet vil også etablere laster, lastkombinasjoner opplagerbetingelser, etasjer, akse og forbindelse mellom elementer. Til slutt vil det utføre en analyse som gir krefter på konstruksjonen i ULS og SLS. Det andre scriptet overfører en armeringsmodell fra FEM-Design til Revit, samt gir mulighet for å overføre strukturelle elementer.

Det eksisterer i dag verktøy som utfører en tilsvarende overføring som scriptene, StruXML. Resultat fra scriptene sammenlignes opp mot dette verktøyet, for å se hvordan scriptene presterer i forhold til dagens muligheter. Til tross for at scriptene har sine begrensninger, eksisterer det fordeler som ikke StruXML kan tilby.

Det er noen forutsetninger for at en overføring skal være vellykket, men disse er minimale sett opp mot manuelt arbeid. Resultatene anses å være nøyaktige og tidsbesparende. Med bakgrunn i dette, konkluderes det at implementering av parametrisk design i en prosjekteringsfase vil bidra til et mer effektivt resultat.

Abstract

The construction industry is undergoing a major technological development that brings along new methods, which can contribute to a more efficient design of structures. Parametric design is an emerging tool that can be used for this purpose.

This thesis deals with how parametric design can be used to contribute to a more efficient design. A visual programming tool, Dynamo, is used to establish a workflow between the drawing program Revit and the FEA- program FEM-Design. The focus will be on the transfer of elements between the programs, where the reinforcement process is the main part.

Based on this, two scripts have been developed. The first script deals with the transfer of an analytical model from Revit to FEM-Design. The script also establishes loads, load combinations, supports, releases, storeys, and axis. Finally, it will perform an analysis that results in forces acting on the structure in ULS and SLS. The second script transfers a reinforcement model from FEM-Design to Revit and provides the possibility of transfering structural elements aswell.

It already exists a tool today that performs a similar transfer as the scripts, StruXML. Thus, results from the scripts are compared to this tool in order to see how the scripts perfom in relation to today's possibilities. Despite the fact that the scripts have their own limitations, they provide som benefits that StruXML cannot offer.

There are some prerequisites for the transfer to be successful, but these are minimal compared to manual work. The results are considered to be accurate and efficient. Based on this, it is concluded that implementation of parametric design in a design will contribute to a more efficient design.

Innholdsfortegnelse

Figurliste

Tabelliste

Begreper

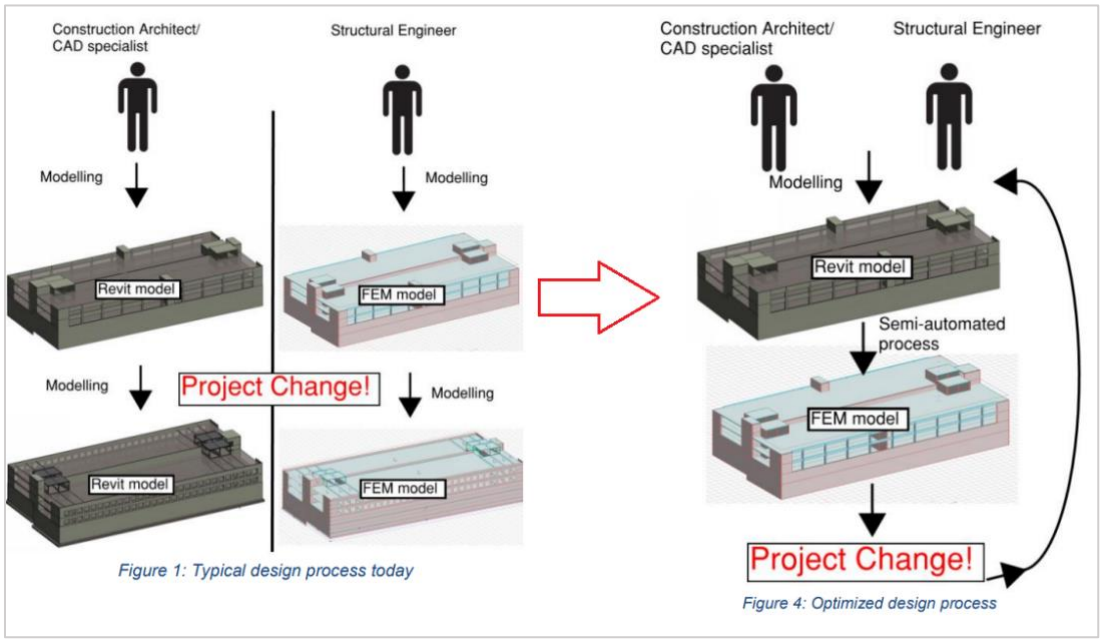
Begrep	Forklaring
Node	En forhåndslaget kode som utfører en spesifikk operasjon
Input for node	Informasjon som gis til en node
Output	Informasjon som sendes ut fra en node
Wire	Brukes for å koble sammen en node til en annen.
Script	Et sett med instruksjoner som utføres av et program
Input for script	Et sett med parabler som kan endres i Dynamo Player
Type	En representasjon av en familie med spesifikke parametriske egenskaper
Strukturelle elementer	Brukes som fellesbetegnelse for strukturelle bjelker, søyler og dekker
Lasttilfelle	Et tilfelle som kategoriserer en last utifra navn, hvilken type last det er og hvor lenge en kraft påvirker konstruksjonen
Lastkombinasjoner	Kombinerinasjon av laster med tilhørende lastfaktorer for å skape en mest ugunstig lastbilde for en konstruksjon
Duration class	Handler om hvor lenge en kraft påvirker en konstruksjon i løpet av dens levetid.
DeadLoad	En konstant last på en konstruksjon som er grunnet egenvekt til et element
Avtrapping	Stegvis reduksjon av armeringsmengde
Polycurve	En kurve bestående av to eller flere kurver koblet sammen
Beregningsmodell	Betegnelse for en modell som er laget i FEM-Design
Fysisk modell	En modell som representerer hvordan bygget fysisk ser ut. Det er en slik modell som lages i Revit
Analytisk modell	Forenklet 3D representasjon av en fysisk modell.

Begreper blir nevnt i oppgaven med kursiv skrift og noder blir nevnt med kursiv og fet skrift.

1. Introduksjon

1.1 Bakgrunn og problemstilling for oppgaven

Byggebransjen er under en enorm teknologisk utvikling som åpner opp for nye metoder som kan bidra til en mer effektiv prosjektering. Det er hard konkurranse blant bedriftene, noe som fører til prispress og lave driftsmarginer (Midttun, 2018). For at bedriftene skal være konkuransedyktige og samtidig redusere kostnader, vil bruk av teknologiske verktøy være til god hjelp. Dagens arbeidsflyt er ofte bestående av prosjekterende som utarbeider en BIM modell og FEM modell hver for seg. En endring i prosjektet må gjøres i begge program og kan resultetere i mye og tidkrevende arbeid (Hejnfelt).



Figur 1.1: Viser dagens arbeidsflyt og hvordan parametrisk design kan optimalisere arbeidsflyten. Hentet fra (Hejnfelt)

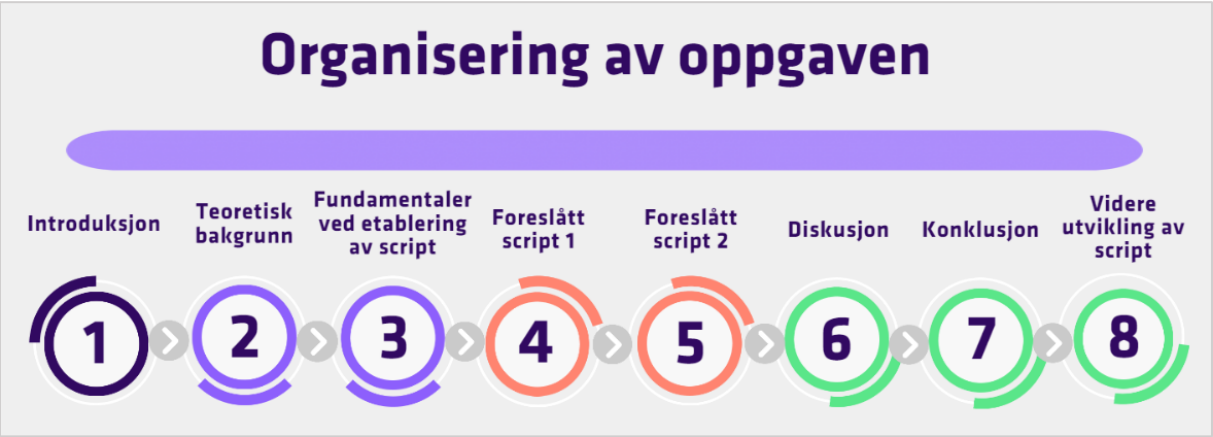
Nye teknologiske verktøy gjør det mulig å introdusere en samhandling mellom tegning- og beregningsmodell, noe som bidrar til en mer optimalisert prosjekteringsfase. Her er parametrisk design en metode som benyttes til dette formålet. Ved bruk av parametrisk design lages modeller som styres av parametere og endres interaktivt, slik at modellen oppdateres automatisk (Lindholm, u.d.). Det er fremdeles et nytt konsept i byggebransjen og ses på som et mulig konkurransefortrinn hos bedrifter.

Bakgrunn for denne oppgaven er å utforske hvordan parametrisk design bidrar til en mer effektiv prosjekteringsfase. Det skal brukes et visuelt programmeringsprogram, Dynamo, for å etablere en effektiv arbeidsflyt mellom tegningsprogrammet Revit og beregningsprogrammet FEM-Design. Her skal det settes søkelys på overføring av elementer mellom programmene, der armeringsprosessen er sentral. Basert på dette er det kommet frem til følgende problemstilling:

- ***Hvordan kan Dynamo etablere en effektiv arbeidsflyt mellom Revit og FEM-Design for en mer optimal prosjektering?***

1.2 Organisering av oppgaven

Oppgaven består av 8 kapitler som er illustrert i *figur 1.2*. Det er også gitt en kort innføring i hva hvert kapittel omhandler.



Figur 1.2 En illustrasjon som viser hvordan oppgaven er delt opp. Laget i (Easel, u.d.)

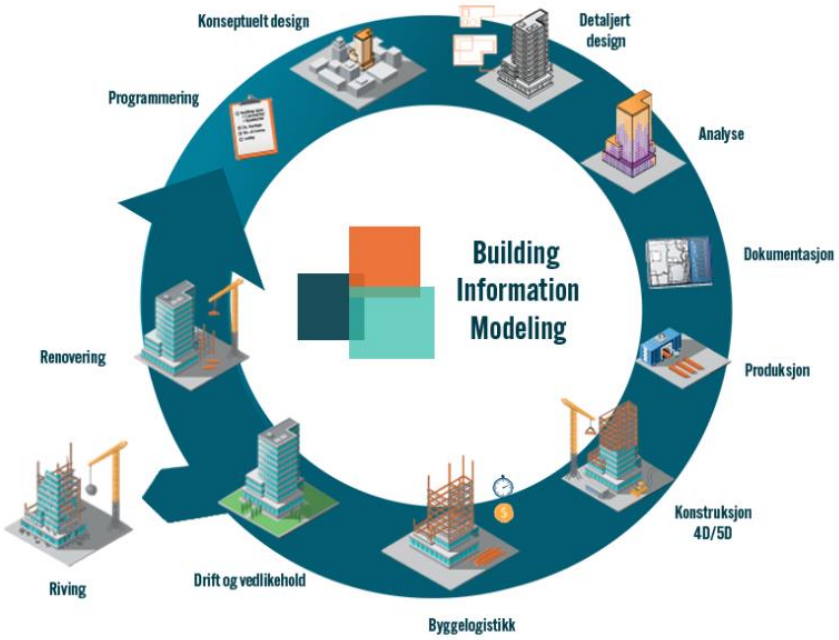
- Kapittel 1 gir en introduksjon og presenterer problemstilling for oppgaven.
- Kapittel 2 inneholder teoretisk bakgrunn for oppgaven.
- Kapittel 3 presenterer fundamentaler ved etablering av script
- Kapittel 4 gir et forslag på første script, som er en overføring av analytisk modell i Revit til FEM-Design.
- Kapittel 5 gir et forslag på andre script, som er en overføring av armeringsmodell i FEM-Design til Revit.
- Kapittel 6 diskuterer forutsetninger og begrensninger av scriptet, eksisterende metoder i dag og hvordan scriptet kan sammenlignes med dem.
- Kapittel 7 gir en konklusjon for oppgaven.
- Kapittel 8 tar for seg videre utvikling av script

2 Teoretisk bakgrunn

Dette kapitellet tar for seg relevant teori for oppgaven, samt en innføring i programvarene som brukes. Som teori for oppgaven gis det en innføring BIM, Paramaterisk design og detaljering av armering i dag og fremtiden. Programvarer som benyttes i oppgaven er Dynamo, Revit og FEM-Design, som gis en kort beskrivelse.

2.1 BIM

Et byggeprosjekt er komplekst og består av et sammarbeid mellom flere aktører som skal levere et prosjekt av høy kvalitet. Det er derfor viktig at aktører er klar over sine egne oppgaver og samhandler mellom hverandre på en god måte. Her er BIM blitt et viktig verktøy som bidrar til en økt effektivitet og kvalitet i byggebransjen. Bygningsinformasjonmodellering (BIM) er en prosess for å etablere og administrere informasjon om et bygg gjennom sin levetid. Som en del av denne prosessen utvikles det en koordinert digital beskrivelse av alle aspekter i et prosjekt, ved hjelp av en samlet 3D modell (Hamil, 2021). Det er viktig å spesifisere at BIM er en prosess som kan deles opp i 4 faser; planlegging, design, bygging og til slutt drift, vist i *figur 2.1.*



Figur 2.1: Viser en grafisk representasjon av Bygnings informasjons modellering. Hentet fra: (NordicBIM, u.d.)

BIM er sentral innenfor den teknologiske utviklingen av byggebransjen på grunn av dens mange fordeler når det gjelder kostnader, sikkerhet og effektivitet. En av de største fordelene med BIM er at byggeprosjektene foregår over kortere tid med mer effekitvitet. Det er en tilnærming som fremmer sammarbeid og kommunikasjon, noe som fører til at alle har tilgang til all informasjon i et prosjekt. Dette gir en bedre oversikt over prosjektet og kan bidra til å redusere svinn, samt legge til rette for en mer nøyaktig og realistisk kostnadestimering (What is BIM (Building Information Modeling), 2022).

Primært er BIM brukt av arkitekter og rådgivere der det lages modeller for å visualisere en konstruksjon. Byggebransjen ønsker også at digitaliseringen tas videre med ut på byggeplassen. Dette er blitt mulig å gjøre gjennom nye verktøy som blant annet BIM kiosker. Dermed trenger ikke håndtverkere å basere seg på tegninger, som kan være utdatert og føre til feilarbeid (Aarhus, 2019). Utførende og prosjekterende vet at endringer og oppdagede feil senere i prosjektet gir store kostnader, noe effektiv bruk av BIM kan redusere risikoen for.

2.2 Parametrisk design

Digitalisering driver frem nye innovasjoner som endrer måten konstruksjoner utformes, bygges, administreres og drives på. I dag forventes det en effektiv prosjektering som muliggjør innovative og komplekse design av konstruksjoner. Her har parametrisk design utviklet seg for å møte dette behovet, og gir fleksible verktøy som tillater ubegrenset kreativitet under design (Allplan, 2020).

Parametrisk design bygger på en algoritme, som er en presis beskrivelse av en serie arbeidsoppgaver, som utføres for å løse et problem (Gillis, 2022). En parametrisk modell definerer et sett med regler og parametere som beskriver modellen og skaper relasjoner mellom informasjon og elementer. Denne måten å modellere på gjør det enklere å håndtere endringer, enn ved tradisjonell 3D modellering. Ofte er det en tidkrevende prosess å utarbeide en parametrisk modell, noe som resulterer i at et det ikke alltid er tidsbesparende. Dermed er det gunstig at en parametrisk modell utvikles slik at det kan også brukes for fremtidige prosjekter.

Selv om en endring i utgangspunktet er på en komponent, kan det påvirke andre relaterte komponenter som kan resultere i en tidkrevende korrigeringsprosess. Ved parametrisk design er konstruksjonen og dens komponenter bygd på relasjoner. Når komponenter er koblet sammen og relasjoner opprettholdes, vil en endring av en enkelt komponent også endre på tilknyttende komponenter. Å eliminere manuelt arbeid bidrar til en økt produktivitet, da modellen lages raskere. En automatisk generering av en modell reduserer også risiko for menneskelige feil som kan føre til kostbare feilberegninger. (Construsoft).

Parametrisk BIM modellering er fremtidens svar på å kunne håndtere mer komplekse konstruksjoner med færre ressurser og strammere tids- og kostnadsbegrensninger. Etter hvert som parametrisk design utvikles, vil nye innovative metoder fortsette å forandre hvordan konstruksjoner designes og bygges. Riktig bruk av parametrisk design kan være en metode for å lage kompleks geometri for konstruksjoner, som ville vært utfordrende å modellere og anaylsere manuelt (Smith, 2020).

2.3 Armeringsdetaljering i dag og fremtiden

Armeringsprosessen i dagens praksis består i hovedsak av to aktører; prosjekterende og utførende. Den prosjekterende er ansvarlig for beregninger og fremstilling av armering som er nødvendig for å ivareta konstruksjonen sin integritet. Disse blir levert til utførende, som er ansvarlig for å utføre arbeidet i henhold til prosjekteringsforutsetninger (Norsk betongforening, 2018). I dagens praksis blir detaljetegninger av armering levert som enten 2D- eller 3D- tegninger. Hvilket detaljenivå som benyttes varierer mellom prosjekt, men det er naturlig å anta at 3D- tegninger blir mer og mer anvendt ettersom byggebransjen er under en teknologisk utvikling (Indovance, 2022).

Detaljetegninger av armering i 3D gir betydelige fordeler i forhold til en 2D-modell. Fra prosjekterende sin side skaper det en bedre oversikt over detaljeringen, noe som gjør det enklere å finne eventuelle feil i en prosjekteringfase. Dersom det oppstår behov for endringer, oppdateres armeringstegninger og tabeller automatisk, slik at det ikke er trengst å gjøre arbeid på nytt. Armeringsmengde blir beregnet automatisk fra modellen, noe som luker vekk personlige feil og bidrar til en mer pålitelig modell. Dette vil resulterer i en mer nøyaktig kostnadsestimering av armering, samt mindre bortkastet materiale på byggeplass. (Allplan, 2021)

Armeringsdetaljering er en viktig del av et byggeprosjekt og involverer et tett sammarbeid mellom prosjekterende og utførende. I hovedsak handler armeringsdetaljering om å lage tegninger som formidler viktig informasjon til utførende på byggeplass. Disse tegningene er til hjelp for at utførende skal vite nøyaktig plassering av armeringsjern i en forskaling før betongen skal etterfylles. (Enweetech, u.d.)

Siden 2D-tegninger er i hovedsak skisser med forklaringer, er det fare for at utførende kan misforstå hva prosjekterende har tenkt. Ved å implemenetere 3D- tegninger på byggeplass minsker denne risikoen. Det er enklere å forstå hvordan armering skal legges når den er visualisert, noe som gjør at utførende slepper å tolke hva prosjekterende har tenkt.

Utifra en 2D-tegning kan det oppstå problemer for utførende ved at armeringsjern kolliderer. En grunn til dette er at tegningen kun gir en planoversikt over geomtrien, og ikke hvor den faktiske geometrien ligger. Ved en visuell representasjon av armering vil det enkelt identifiseres om det oppstår en kollisjon mellom armeringsjernene.

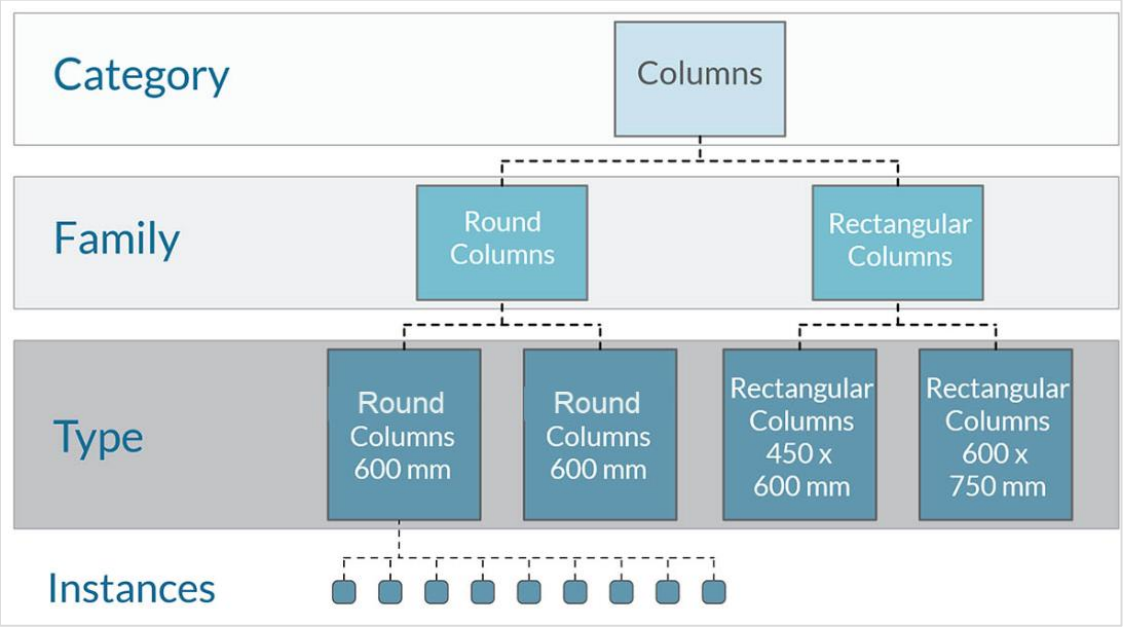
Ofte er problem som oppstår på byggeplass grunnet mangel på kommunikasjon, forståelse og sammarbeid mellom prosjekterende og utførende. Implementering av 3D tegninger kan bidra til et mer nøyaktig og effektiv prosjekt, for både prosjekterende og utførende (Tekla).

2.4 Revit

Revit er et 3D-modelleringsprogram som brukes til å prosjektere, designe, visualisere og koordinere et byggeprosjekt. Programmet tilrettelegger for et godt tverrfaglig samarbeid mellom alle aktører i et byggeprosjekt (nti, u.d.). Revit har en tilgjenglig API som muliggjør at andre applikasjoner får tilgang til data og kan samhandle med Revit sine komponenter. API er et akronym for «Application Programming Interface» og er en fellesbetegnelse på hvordan ulike programmer snakker sammen (Wyatt, u.d.). Dynamo er et eksempel på programmer som utnytter seg av Revit sin API.

2.4.1 Element hieraki i Revit

Hvordan et element er bygget opp i Revit, kan forklares utifra den hierarkiske strukturen til et Revit element. Enkelt forklart kan hierarkiet brytes opp i *Categories, Families, Types* og *Instance*. *Figur 2.2* viser hvordan hierarkiet er for en søylekategori.

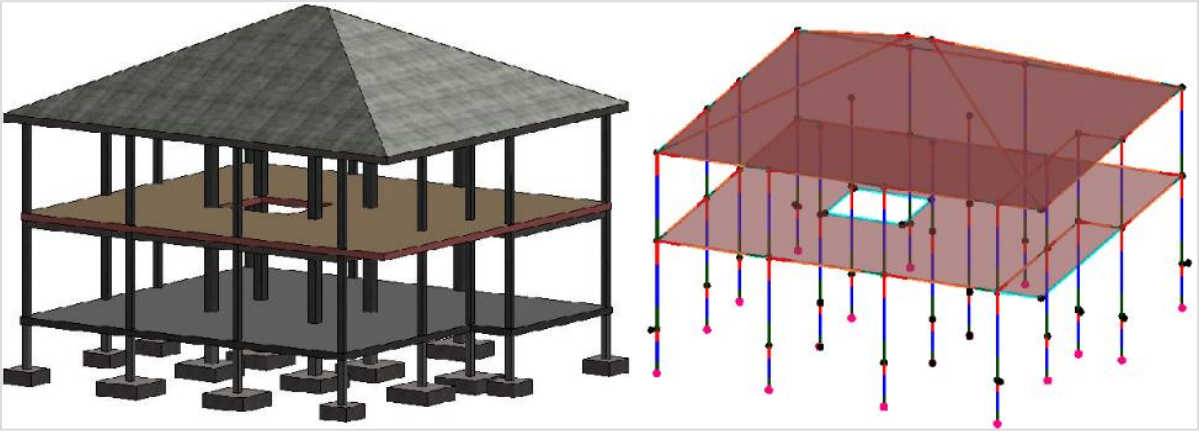


Figur 2.2: Grafisk represetnasjon av hvordan Revit hierarkiet er bygd opp. Hentet fra: (Dynamo Primer, 2019)

På toppen av hierarkiet er *kategorier* som kontrollerer organisasjonen og grafisk representasjon av familier i en modell. Kategorier kan brytes ned til *familier* som defineres som en gruppe elementer med felles egenskaper og grafisk representasjon. Familier kan videre deles opp i *typer* som er en representasjon av en familie med spesifikke parametriske egenskaper. På bunnen av hierarkiet er et *tilfelle* som er en individuell representasjon av en type og består av unike parametriske egenskaper (Knittle).

2.5 FEM-Design

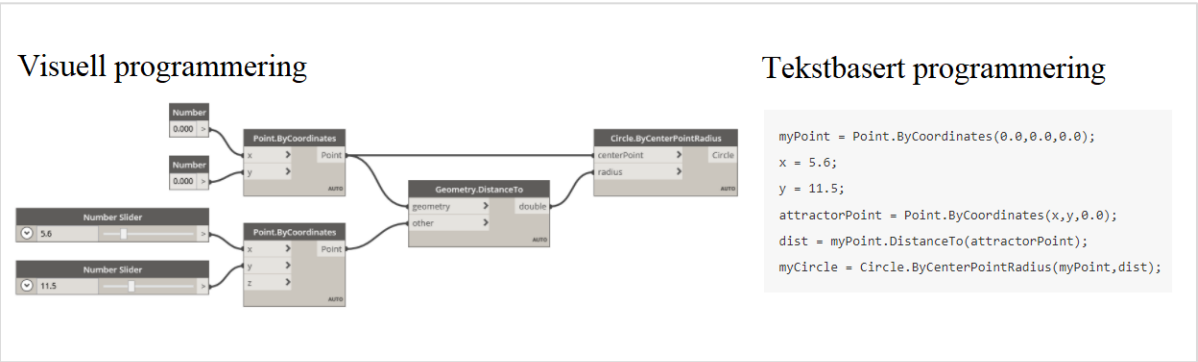
FEM-Design er en programvare som er utviklet av Strusoft. Det er et avansert *finite element method* (FEM) program som brukes til modellering, analyse og design av konstruksjoner. Det baserer seg på FEM teori og er et verktøy for å beregne tilnærmert løsninger for komplekse matematiske problemer. Ved modellering i FEM-Design lages det en analytisk modell som er en forenklet representasjon av en fysisk konstruksjonsmodell, som vist *figur 2.3*. FEM-Design har et API gjennom XML-formatet StruXML som også er tilgjenglig for eksterne programmeringsverktøy, som Dynamo (Strusoft, u.d.).



Figur 2.3: Fysisk modell til venste og analytisk modell til høyre. Hentet fra: (Azevedo, 2014)

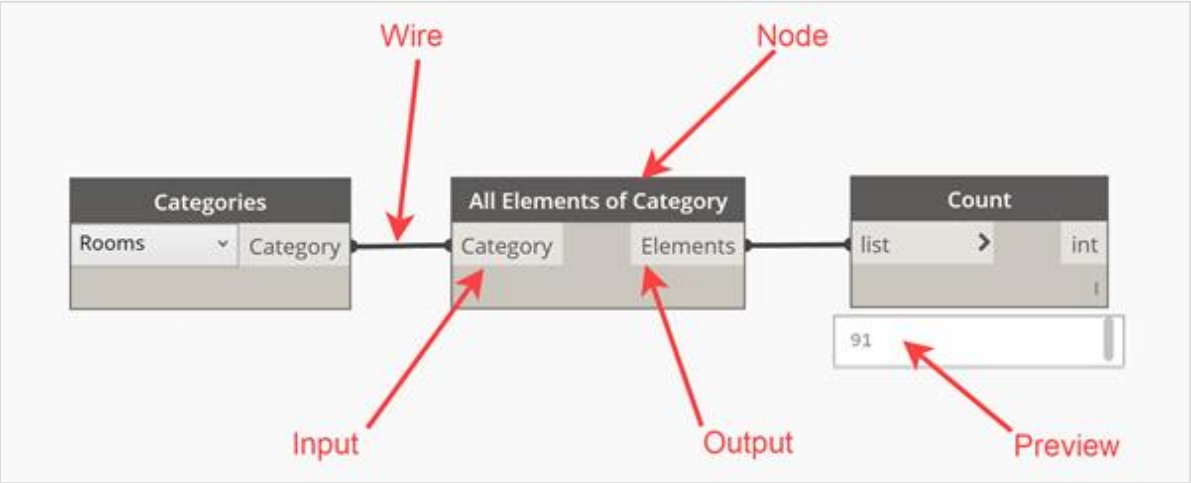
2.6 Dynamo

Dynamo er et visuelt programmeringsprogram for Revit. Programmering handler om å sette sammen et sett av instruksjoner som forteller datamaskinen hvordan den skal utføre en spesifikk oppgave (Jalli, 2022). Dette er essensen i både visuell- og tekstbasert programmering. Det som skiller dem er hvordan instruksjonene er presentert, enten basert på tekst eller grafiske elementer, som illustrert i *figur 2.4*. Siden det ikke er nødvendig med forkunnskaper om programmering for å bruke Dynamo, er det en attraktiv form for programmering for ingeniører. (Asti, u.d.)



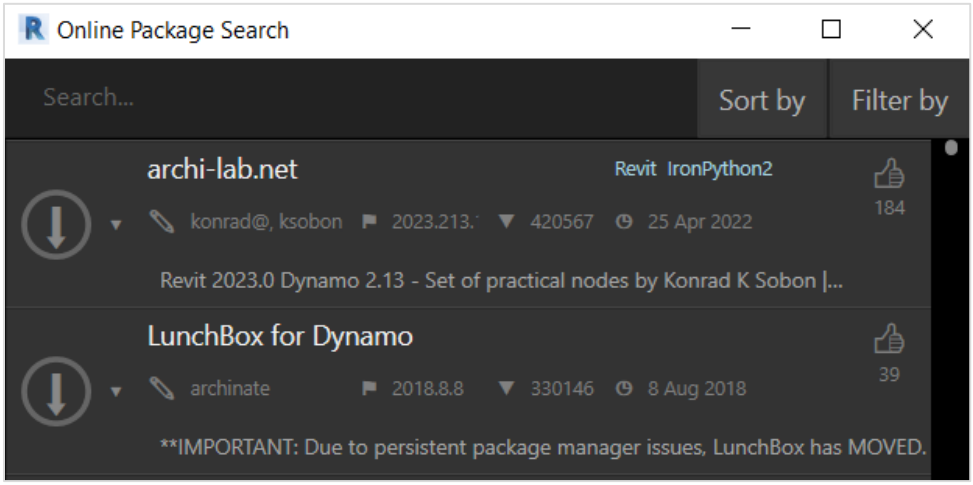
Figur 2.4 Forskjellen mellom visuell- og tekstbasert programmering (Dynamo Primer, 2019).

Istedenfor å håndtere data ved hjelp av koding, brukes Dynamo til å manipulere grafiske elementer kalt «*Noder*». En node utfører en spesifikk oppgave og er bygd opp av «*inputs*» og «*outputs*», som kan kobles opp til hverandre ved en «*wire*». En *wire* er tilkoblet *output* hos en node og *input* til en annen node og skaper et forhold mellom nodene. Sammensetningen av flere noder og wires skaper et nettverk som representerer nødvendige steg for å gjennomføre ønsket produkt, og er det som utgjør et «script» (Dynamo Primer, 2019).



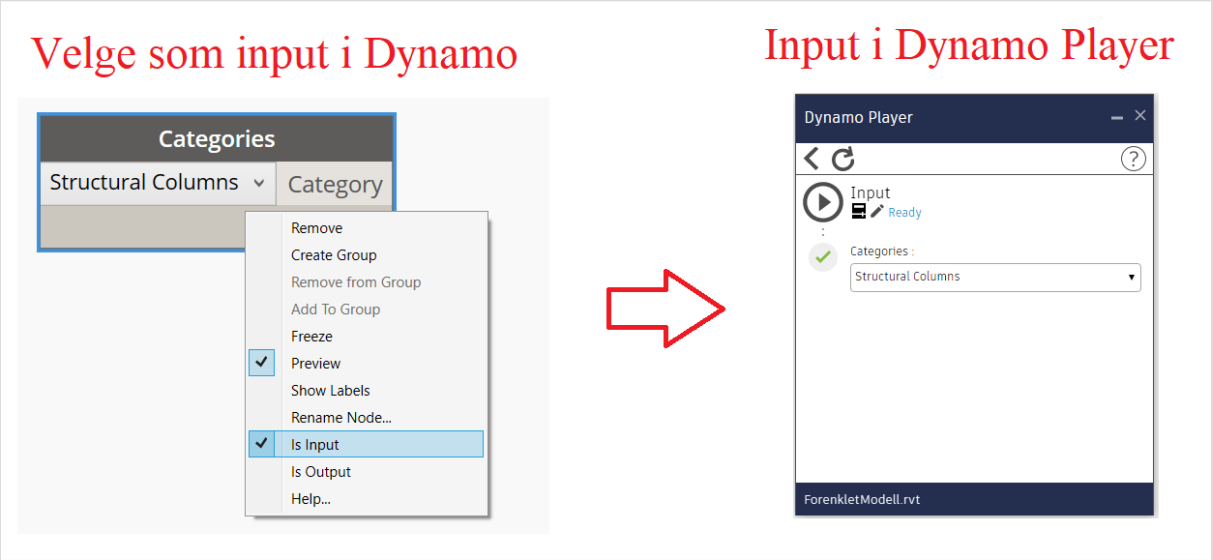
Figur 2.5: Viser hvordan noder er bygd opp i Dynamo og hvordan wires sammenkobler dem (Brito, 2019).

Dynamo er et «open-source community», bygget på et samfunn som bidrar til å videreutvikle og forbedre programmet. Dette gjør at enhver bruker kan lage og publisere sine egne pakker med noder, som også er tilgjenglig for andre brukere. Disse pakkene kan finnes i et bibliotek, som har en oversikt over alle tilgjenglig pakker for Dynamo. (Kilkelly, 2018)



Figur 2.6: Dynamo bibliotek, hvor nodepakker kan lastes ned.

For å tilrettelegge for et brukervennlig script bør alle variable parametere i scriptet settes som «*inputs*». Dersom en node er definert som en *input* vil brukeren ha muligheten til å endre denne i «Dynamo Player», som er en egen add-in i Revit. Dynamo Player gir en oversikt over alle *inputs* i scriptet, samt gjør det mulig å endre disse, og til slutt kjøre scriptet uten å åpne det i Dynamo.



Figur 2.7 Viser hvordan en node kan settes som input og hvordan Dynamo Player ser ut

3 Fundamentaler for etablering av script

Før det etableres et script bør det gjøre seg noen tanker om hva som er ønskelig å oppnå og hvordan komme frem til målet på en effektiv måte. Ved scripting er det utallige metoder for å komme frem til et resultat. I hvor stor grad et script kan betegnes som en effektiv løsning vil dermed være subjektivt og avhengig av forventninger til scriptet.

Etter hvert som et script blir større og mer komplekst, blir det stadig mer uoversiktlig. Det blir utfordrende å holde styr på hva og hvor spesifikke operasjoner utføres, finne feil når ting går galt og hvor utvidelse av script skal legges til. For å opprettholde en god oversikt i scriptet er det lurt å etablere en organiseringsstrategi i forkant av prosjektet.

Dette kapitellet tar for seg fundamentaler ved utarbeiding av et script

3.1 Innhenting av informasjon

For å oppnå et godt resultat er det nødvendig å stille seg kritisk til kilder som benyttes. Gjennom oppgaven er det hentet informasjon for den teoretiske delen, samt for å opparbeide kunnskap rundt programvarene.

3.1.1 Dynamo

Hovedprogrammet for oppgaven er Dynamo, der relevant kunnskap er opparbeidet gjennom prøving og feiling, forum og hjelp fra veileder i Norconsult. Å utarbeide et script er en tidkrevende prosess, der flere løsninger må utforskes. Dersom en løsning ikke anses som god nok blir denne fokastet og en ny metode forsøkt.

Når det ikke har finnes løsning med prøving og feiling, har hjelp fra veileder og Dynamo sitt forum vært til stor hjelp. Dynamo har et åpent forum, hvor brukere kan publisere problem og løsninger. Det er ofte andre brukere som har hatt en tilsvarende problemstilling, slik at det allerede kan ligge ute en løsning.

3.1.2 Pålitelighet i oppgaven

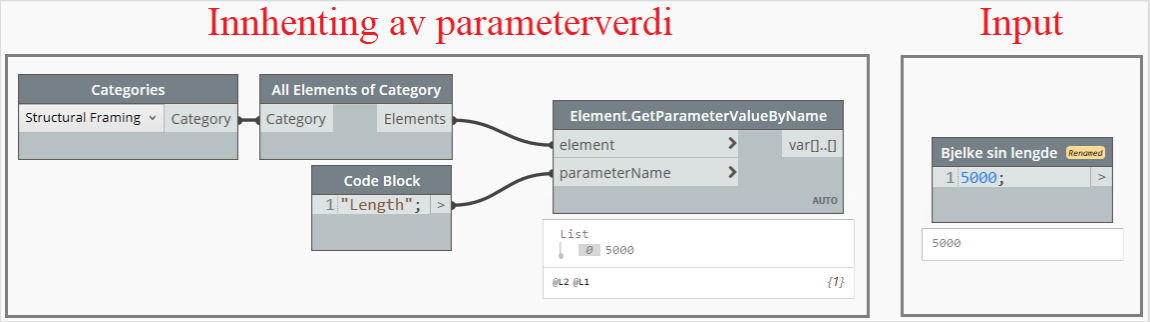
For at et script skal være av verdi, er det nødvendig at det produserer et resultat som kan stoles på. Dette oppnås med at scriptet kvalitetsjekkes kontinuerlig gjennom oppgaven. Ved endringer og nye operasjoner i scriptet, testes scriptet for å være sikker på at resultat er som ønsket. For å kvalitetsikre er også resultat sammenlignet opp mot esksisterende verktøy som tilbyr tilsvarende operasjon, gjennomgått i *delkapittel 6.3*.

Det er benyttet et forum for å finne løsninger på utfordringer. Her kan hvem som helst publisere, og det er nødvendig å være kritisk til informasjon som finnes. Eventuelle forslag testes ut og forkastes dersom de ikke er til hjelp for scriptet. Dermed vil et velfungerende script være en god indikasjon på at kildene som brukes er pålitelige.

3.2 Minimere antall inputs

For å skape et brukervennlig og tidsbesparende script er det ønskelig å minimere antall inputs i scriptet. Dette resulterer i mindre arbeid for bruker, noe som også vil redusere faren for personlige feil. For å kunne redusere antall inputs i et script bør det finnes måter å uttrykke sammenhenger på, istedenfor å lage nye inputs.

En metode som brukes mye ved scripting er innhenting av parameterverdier. Et eksempel på dette er at en bjelkes lengde kan finnes på to måter, som vist i *figur 3.1*. Ved innhenting av parameterverdier kreves ingen input, men det må brukes 4 noder for å oppnå dette. Den andre metoden er å bruke 1 input og dermed er det kun nødvendig med en node. En ulempe med innhenting av parameterverdier er at det fører til et tyngre script, da flere noder må brukes



Figur 3.1: Innhenting av parameterverdier istendefor å bruke input

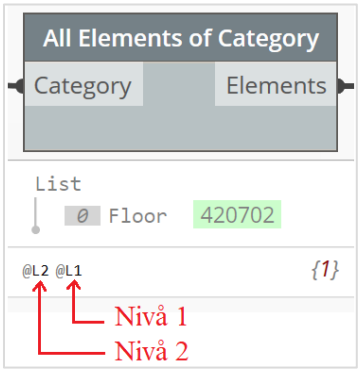
En annen metode er å bruke *Code Blocks* for å lage formler. Ved å uttrykke en input ved hjelp av en annen node kan antall input reduserest. En bakside med dette er at mange prosesser blir avhengig av hverandre, noe som kan resultere i et tyngre og mer komplisert script. En liten endring eller feil i scriptet kan føre til store problemer. Dermed må det gjøres en vurdering om hva som er best for scriptet av færre inputs eller et raskere script

Figur 3.2: Viser hvordan code blocks kan brukes for å lage formler

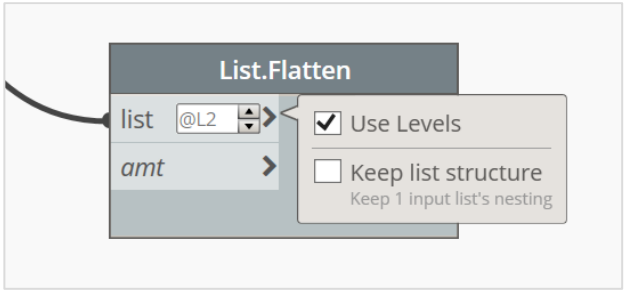
3.3 Lister i Dynamo

For å kunne mestre Dynamo på en god måte er det essensielt å kunne håndtere arbeid relatert til lister. En liste er en samling av flere elementer og handler om hvordan data organiseres. Når data blir større og mer komplekst, er det enklere å håndtere en samling av elementer istendefor et og et element. Dette gjør at lister ofte brukes for å samle elementer med parametriske relasjoner. Når det jobbes med flere elementer samtidig, kan listene ofte bli komplisert og det er enkelt å miste kontroll. Derfor er det viktig å etablere en god listestruktur tidlig i scriptet.

Lister kan være flerdimensjonale, noe som vil si at en liste kan inneholde en annen liste. Derfor deler Dynamo inn listene sine i nivåer som viser dimensjonene til en liste, vist i *figur 3.3*. Når det arbeides med lister, dukker det opp situasjoner der man ønsker å bruke et spesifikt nivå av en liste. Dette kan enkelt gjøres i de fleste noder, der det hukes av «Use Levels» og velge ønsket nivå, vist i *figur 3.4*. Her kan også «Keep list structure» hukes av, som sørger for at eksisterende listestuktur opprettholder.

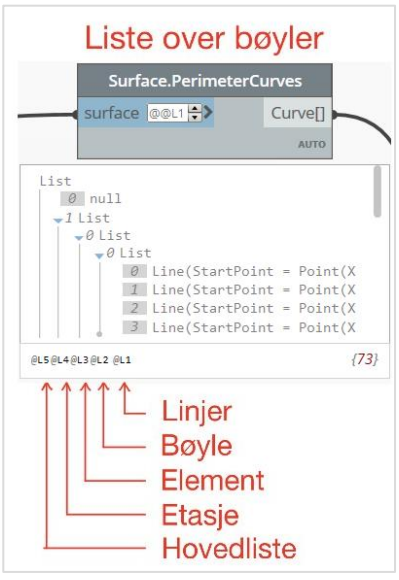


Figur 3.3: Nivåer i en liste



Figur 3.4: Velge nivå i en liste

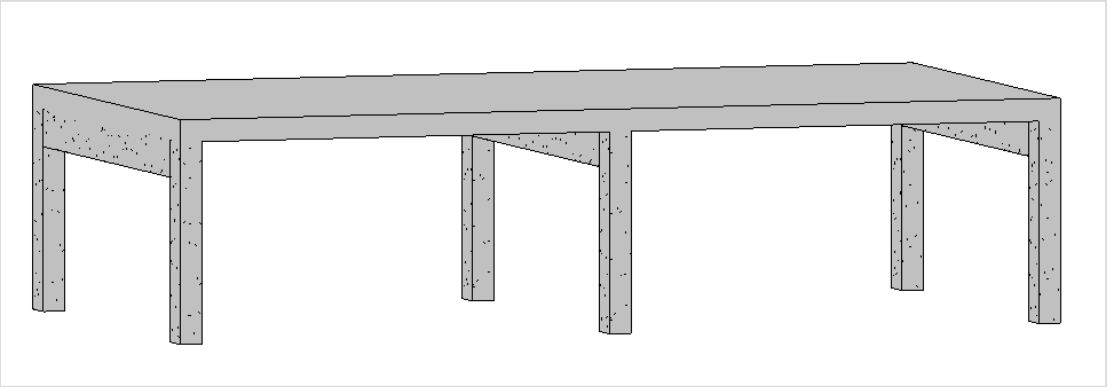
Et eksempel på hvordan en liste kan se ut er gitt i *figur 3.5*, som viser hvordan bøylearmering for en søyle. Det øverste nivået er alltid en hovedliste, som deles opp til neste nivå som i dette tilfelle er etasjer. Videre deles denne opp i elementer, som gir en oversikt om alle søyler som eksisterer innenfor denne etasjen. Deretter deles denne listen opp i bøyler som forteller hvor mange bøyler som eksisterer i hver søyle. Til slutt deles listen opp igjen for å finne linjer som skaper geometrien til en bøyle.



Figur 3.3: Liste over nivåer for en bøyle

3.4 Forenklet modell for oppgaven

For å lage et utgangspunkt for oppgaven er det laget et enkelt system av 6 søyler, 3 bjelker og et dekke. Beregninger er ikke et sentralt tema for og vil dermed ikke bli utført i denne oppgaven.



Figur 3.4: Viser forenklet modell i FEM-Design som er utgangspunkt for oppgaven

3.4.1 Laster

For å kunne utføre en analyse i FEM-Design er det nødvendig å etablere laster som virker på konstruksjonen. Siden laster er forskjellig og ikke alltid opptrer samtidig, er det nødvendig å skille dem. Ofte skilles det mellom permanente og variable laster. Permanente laster virker på konstruksjonen gjennom tilnærmet hele levetiden, som for eksempel egenvekten til en konstruksjon. Variable laster varier over tid og kan være for eksempel møbler og mennesker i et kontorbygg. (Rajput, u.d.)

For denne konstruksjonen er det påkjent en egenlast fra konstruksjonen, samt en nyttelast på 5 kN/m^2 som virker jevnt fordelt over dekke. Egenvekt er en last som ikke vil endres gjennom levetiden til konstruksjonen, og er dermed en permanent last. Nyttelasten kan variere over tid og vil dermed være en variabel last.

Etter lastene er definert, kombineres de for å skape en mest mulig ugunstig lastsituasjon. Dette gjøres ved å gi hver last sin egen lastfaktor. «*Limit state design*» krever at en konstruksjon tilfredstiller kriterier i både bruddgrense og bruksgrense. Bruddgrense er assosiert med kollaps eller svikt, og forteller om maksimal last en konstruksjon eller et konstruksjonselement kan utsettes for uten å svikte. Bruksgrense handler om å sikre at konstruksjonen er komfortabel og brukbar for menneskelig bruk (Cornelius, 2018). Lastfaktorene som er brukt i bruksgrense og bruddgrense er vist i *tabell 1.*

Tabell 1: Viser lastfaktorer som er brukt i oppgaven

Lasttype	Lastfaktor ULS	Lastfaktor SLS
Egenlast	1,35	1
Nyttelast	1,5	1

4 Forslag til script 1 - Overføring av modell fra Revit til FEM-Design

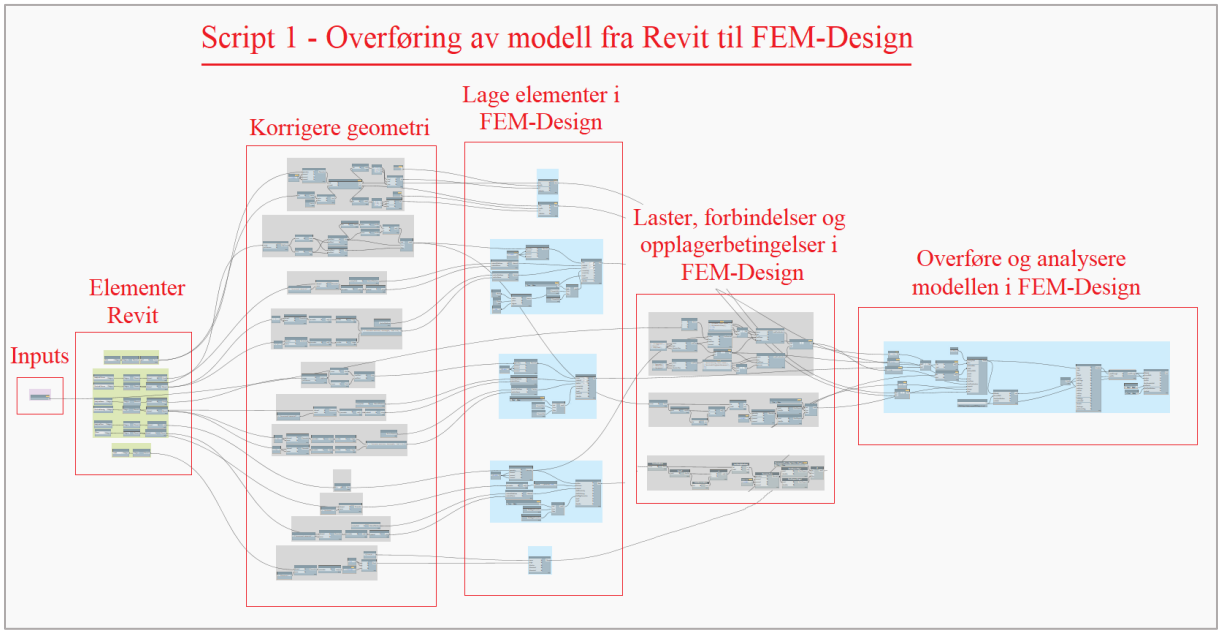
I delkapittel 3.3 er det etablert en forenklet modell som er utgangspunktet for dette scriptet. Scriptet har som formål å automatisk overføre en analytisk modell fra Revit til FEM-Design. Videre skal det etablere laster, lastkombinasjoner, opplagerbetingelser, akser, etasjer, forbindelser og til slutt utføre en beregning av modellen.

4.1 Oversikt over script

Etter hvert som et script vokser og blir mer komplisert, er det nødvendig med en god organisering av scriptet. Dersom det oppstår et behov for å gjøre en endring i scriptet er det viktig at det er enkelt å finne frem. En metode for å oppnå en god organisering er gruppering funksjonen til Dynamo. Denne brukes for å samle noder som utfører en bestemt operasjon. Gruppene kan videre organiseres med fargekoder og en tittel som forklarer hva operasjonen er. Fargekodene som er brukt i scriptet er:

- Operasjoner som henter eller overfører elementer til Revit gis grønn farge.
- Operasjoner som omhandler korrigering av geometri gis grå farge.
- Operasjoner som henter eller overfører elementer til FEM-Design gis blå farge.

I *figur 4.1* er det presentert en oversikt over hvordan scriptet er bygget opp.

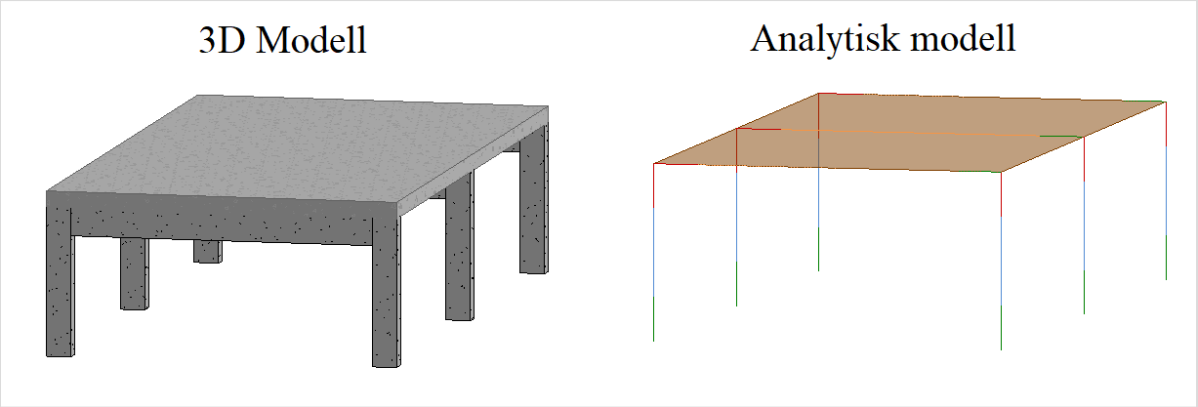


Figur 4.1: Oversikt over script 1

Videre blir scriptet gjennomgått i mer detalje. For å unngå repeterende forklaringer er det noen prosesser som ikke blir presentert, da den er tilsvarende som en annen.

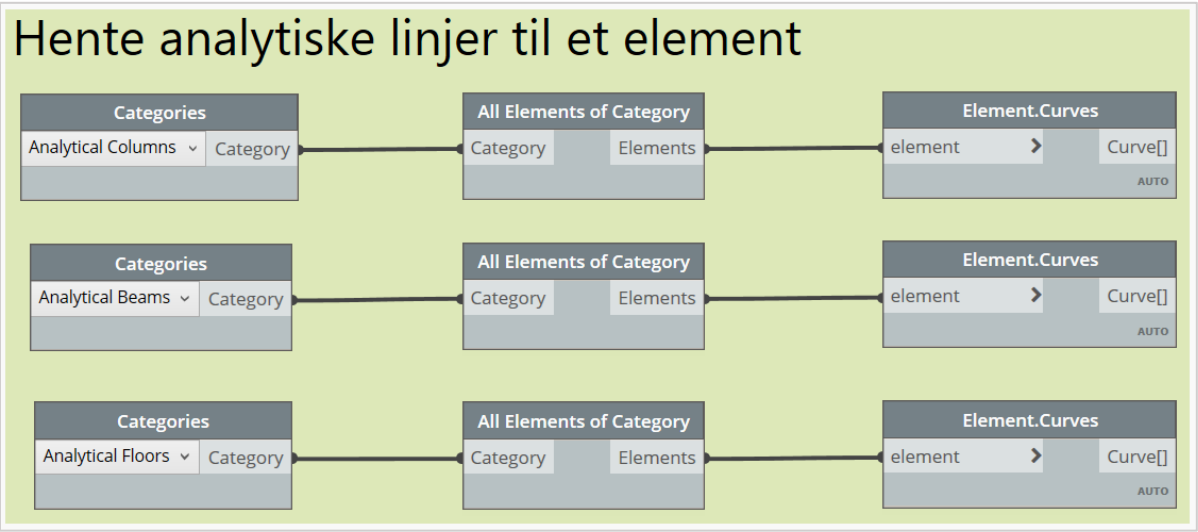
4.2 Analytisk modell

Det er viktig å skille mellom hvordan modellering av elementer foregår i Revit og FEM-Design. Ved modellering i Revit er det ønskelig at elementer representerer hvordan konstruskjonen fysisk ser ut. Da FEM-Design er et beregningsprogram er det viktig elementer blir representert utifra effekten de har på konstruksjonen, og består av en analytisk modell. Derfor er det viktig at de analytiske linjer blir overført på en korrekt måte, slik at beregninger ikke blir feil.



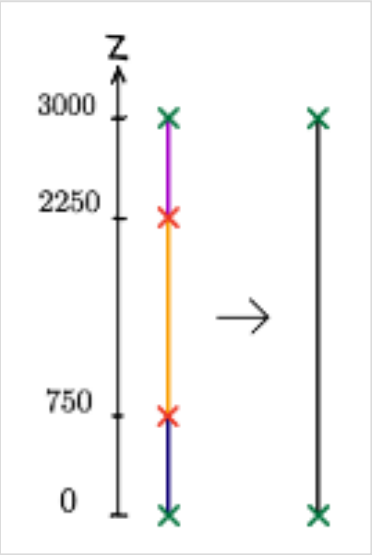
Figur 4.2: Forskjell på en 3D-modell og en analytisk modell i Revit

For å hente analytiske elementer fra Revit brukes noden ***Categories***, som gir Dynamo tilgang til alle kategorier som eksisterer i en Revit modell. Her velges analytiske bjelker, søyler og dekker. Videre brukes ***All Elements of Category*** som henter alle eksisterende elementer innenfor tilhørende kategori. Hvert element har sin unike elementID, noe som gjør det mulig å skille mellom elementer. Til slutt kan ***Element.Curves*** brukes for å hente analytiske linjer til et element. Denne prosessen er vist i *figur 4.3.*



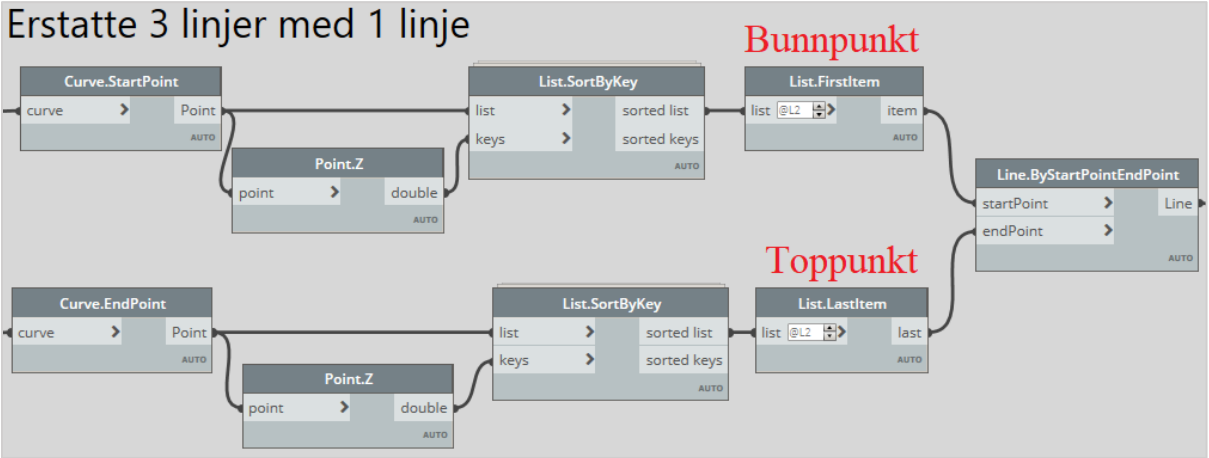
Figur 4.3: Hente analytiske linjer fra bjelker og søyler

Når analytiske linjer hentes fra Revit, oppstår det 3 linjer som representerer den faktiske analytiske linjen. Siden det kun er behov for den faktiske analytiske linjen, må det lages en ny linje basert på startpunkt til første linje og sluttpunkt til siste linje, som vist i *figur 4.4*.



Figur 4.4 Tre linjer til en linje

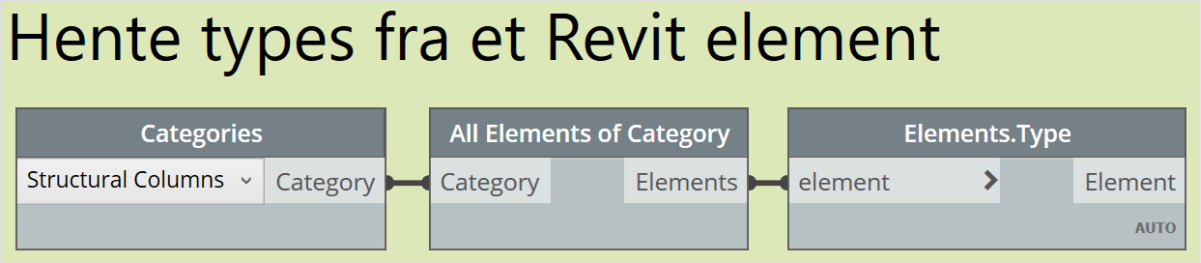
Figur 4.5 viser hvordan dette gjøres i Dynamo. ***Curve.StartPoint*** brukes for å hente alle startpunkt til en linje. Siden det er 3 linjer som representerer den analytiske linjen, vil denne noden også gi 3 startpunkt. For å sikre at riktige startpunkt hentes, er punktlisten sortert etter stigende Z-kordinater. Deretter blir startpunkt med lavest Z-koordinat for hvert element hentet med ***List.FirstItem***. Denne prosessen repeteres med ***Curve.Endpoint*** for å finne sluttpunktene også. ***Line.ByStartPointEndPoint*** brukes til slutt for å trekke en linje mellom disse punktene.



Figur 4.5: Erstatte 3 linjer med 1 linje

4.3 Egenskaper til et element

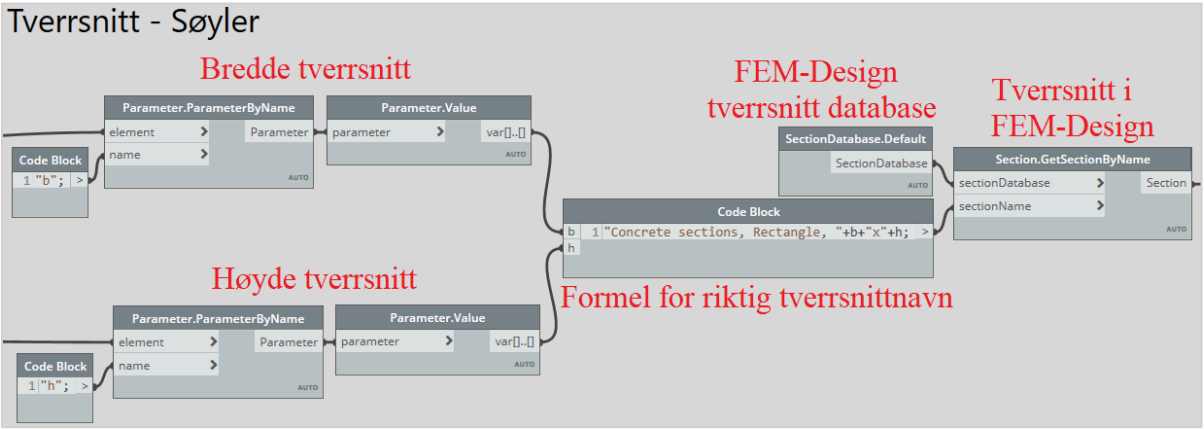
Videre er det nødvendig å definere egenskaper til et element. For et *bar element* oppgis tverrsnitt og betongkvalitet. Dette er informasjon som eksisterer i Revit modellen, slik at Dynamo kan brukes for å hente den ut. Et tverrsnitt er definert som et element i Revit, men selve dimensjonene til tverrsnittet er en *type*. Derfor brukes ***Elements.Type*** slik at Dynamo får tilgang til alle *types* i et element, vist i *figur 4.6*.



Figur 4.6: Viser hvordan Dynamo kan brukes for å hente alle types til et element

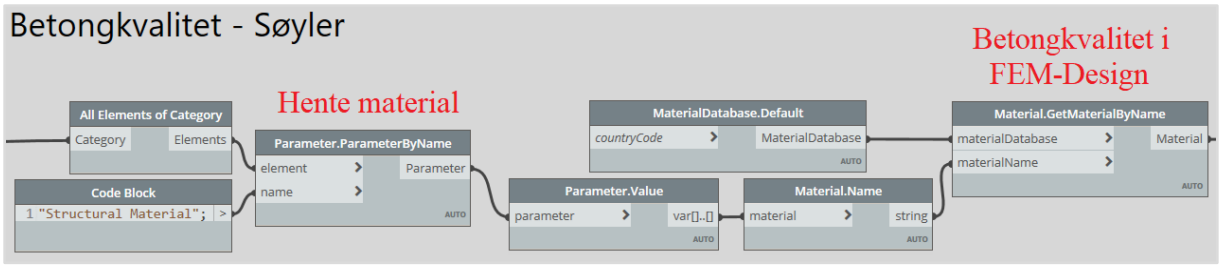
For å kunne hente ut en parameter fra en *type* brukes ***Parameter.*ParameterByName**. Her finnes bredden og høyden til tverrsnittet. For at FEM-Design skal kunne bruke denne informasjonen, er det nødvendig at tverrsnittet som oppgis samsvarer med FEM-Design sin tverrsnitt database.

For et 300x500 *bar element* i FEM-Design er tverrsnittnavnet «Concrete sections, Rectangle, 300x500». Siden tverrsnittet kan endre seg, lages det en funksjon som gjør at navnet alltid endrer seg interaktivt med tverrsnittet, vist i *Code Block i figur 4.7*. Til slutt brukes ***Section.GetSectionByName*** for å lage tverrsnittet i FEM-Design.



Figur 4.7: Viser hvordan tverrsnitt og betongkvalitet blir hentet fra Revit og tilegnet et element i FEM-Design

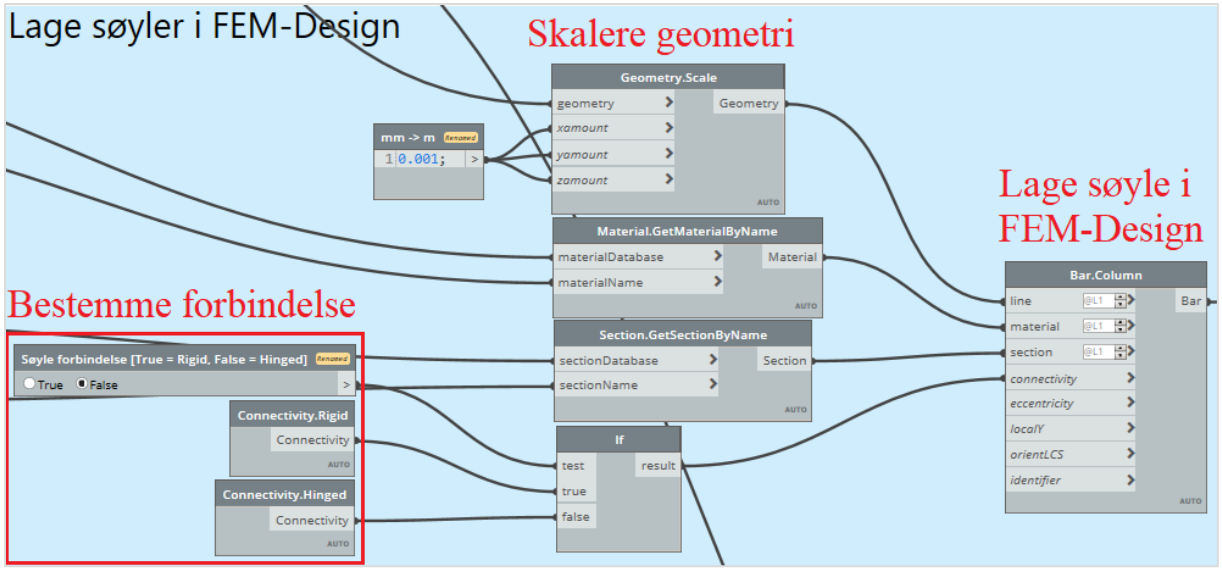
Siden material er et element og ikke en en *type*, trengst ikke elementes å brytes opp til *types* for å finne betongkvaliteten. Tilsvarende som for tverrsnitt, er det nødvendig at navnet på betongkvaliteten stemmer overens med Material databasen i FEM-Design, før *Material.GetMaterialByName* brukes for å gi elementet en betongkvalitet, vist i *figur 4.8.*



Figur 4.8: Viser hvordan betongkvalitet i FEM-Design lages

4.4 Lage søyler i FEM-Design

Da er nødvendig informasjon klargjort for å kunne lage en søyle i FEM-Design, vist i *figur 4.9*. Siden FEM-Design opererer i meter og Revit i millimeter, må geometrien skalerest med ***Geometry.Scale***. For å bestemme endeforbindelsene til søylen er det gitt et valg mellom *hinged* og *rigid.* Når det finnes to valg, kan ***If*** brukes for å lage en test med *true/false* alternativer. Denne testen styres med en ***Boolean*** node, der *True* vil gi en *rigid* endeforbindelse, mens *false* vil gi en hinged endeforbindelse.



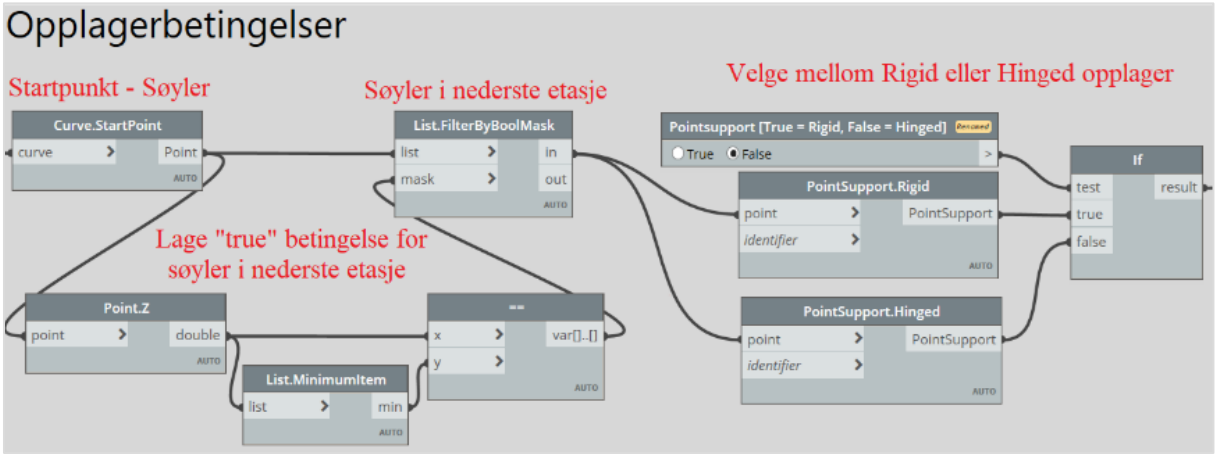
Figur 4.9: Viser hvordan søyler blir overført fra Revit til FEM-Design

Fremgangsmåten er tilnærmet lik for bjelker, søyler og dekke, derfor er ikke prosessen for bjelker og dekker gjennomgått. Likevel nevnes det at et dekke er avhengig av en *surface* istedenfor en *curve* og en tykkelse istedenfor et tverrsnitt.

4.5 Lage opplagerbetingelser i FEM-Design

For å utføre en analyse av en modell er det nødvendig at opplagerbetingelser er definert, som gjøres med Dynamo i *figur 4.10*. Disse plasseres utifra bunnen av en søyle, som kan finnes med å bruke ***Curve.StartPoint*** på en søyle sin analytiske linje. Siden en modell kan bestå av søyler i flere etasjer, må det ta hensyn til at opplagerbetingelser kun plasseres i nederste etasje. Dette oppnås med å finne søylene med lavest Z-koordinat og separere disse fra resten.

Point.Z gir Z-koordinat til bunnpunktet og ***List.MinimumItem*** finner bunnpunkt med lavest Z-koordinat. Disse kobles til ***==*** for å lage en true/false test, der *true* verdier er søyler i nederste etasje. Deretter benyttes ***List.FilterByBoolMask*** for å skille søyler i nederste etasje fra resterende etasjer. Punktene benyttes videre i ***PointSupport.Rigid*** og ***PointSupport.Hinged***. Ved å koble disse til en ***If***, kan brukeren velge mellom en *Rigid* eller *Hinged* opplagerbetingelse.

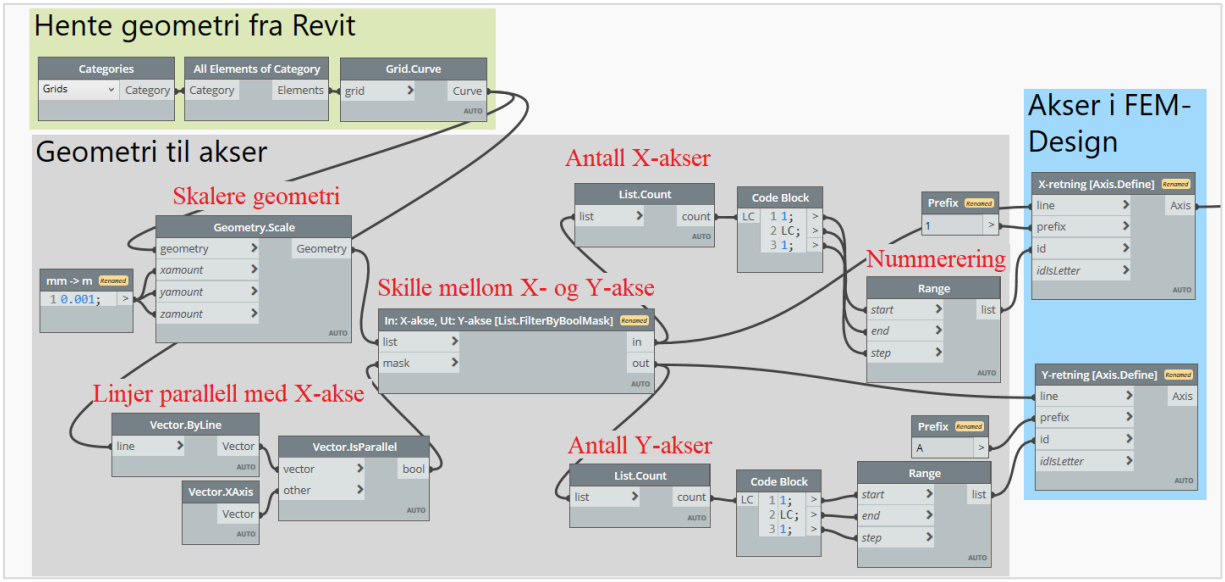


Figur 4.10: Viser hvordan Dynamo kan brukes for å plassere ut Opplagerbetingelser7

4.6 Overføring av akser og etasjer

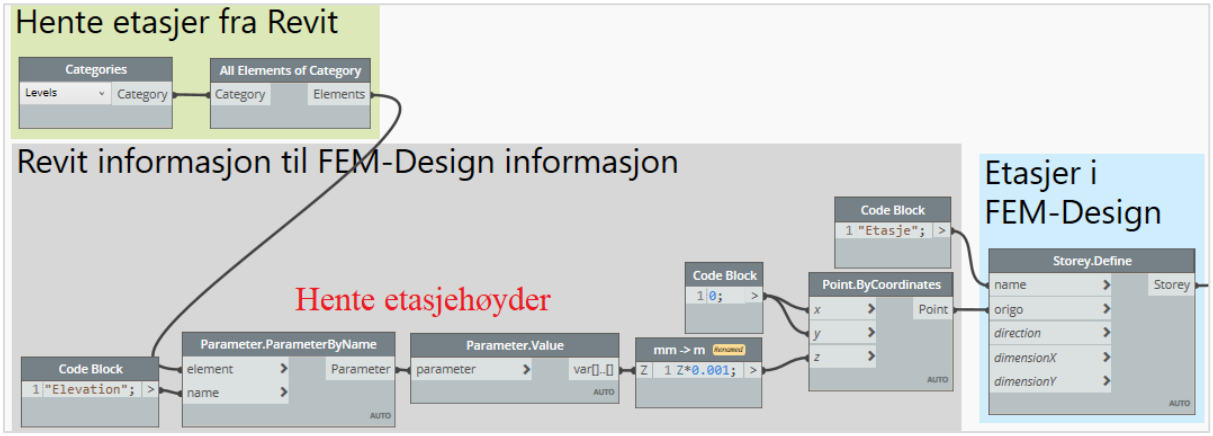
En Revit modell består ofte av akser og etasjer som bidrar til å organisere modelleringen. Dette benyttes også ved modellering i FEM-Design, noe som gjør en overføring gunstig. *Figur 4.11* viser hvordan akser overføres ved bruk av Dynamo. Akser er definert som en kategori i Revit og må derfor brytes ned til et elementer for å hente underliggende geometrien til en akse med ***Grid.Curve***.

For å skille mellom akser i X- og Y retning brukes ***Vector.ByLine*** som gir hver akselinje sin egen vektor. Deretter benyttes ***VectorIsParallell*** som sjekker om en akselinje er parallell med X aksen, som gis med ***Vector.XAxis***. Dette vil resultere i en liste med *true* og *false* verdier som brukes som betingelse i ***List.FilterByBoolMask***. Da er det etablert en liste med akser i X-retning og en liste med akser i Y-retning. ***List.Count*** vil da gi antall akser i hver retning og blir input for ***Range*** som lager en liste med nummerering fra 1 til antall akser. Videre må aksene navngis før ***Axis.Define*** lager akser i FEM-Design.



Figur 4.11: Overføre akser fra Revit til FEM-Design

I likhet med akser, er etasjer en kategori som brytes ned til elementer. Høyden til hver etasje hentes fra et element sin parameter med ***Parameter.ParameterByName*** og brukes som Z-koordinat i ***Point.ByCoordinates***. Disse punktene representerer da høyden til hver etasje som eksisterer i Revit. Punktene kobles videre på ***Store.Define*** som lager etasjer i FEM-Design, vist i *figur 4.12*.

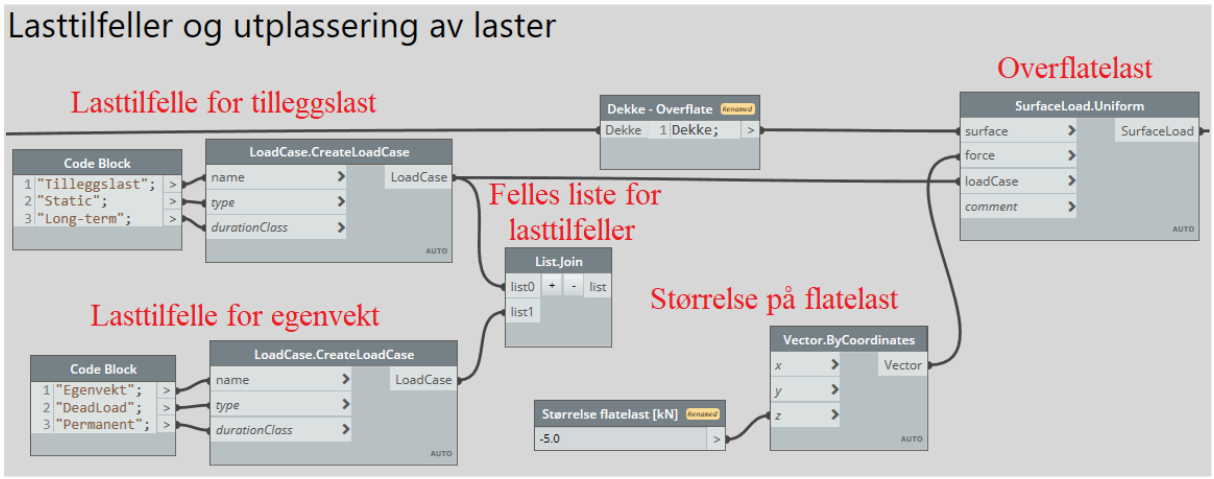


Figur 4.12: Overføre etasjer fra Revit til FEM-Design

4.7 Opprette laster og lastkombinasjoner

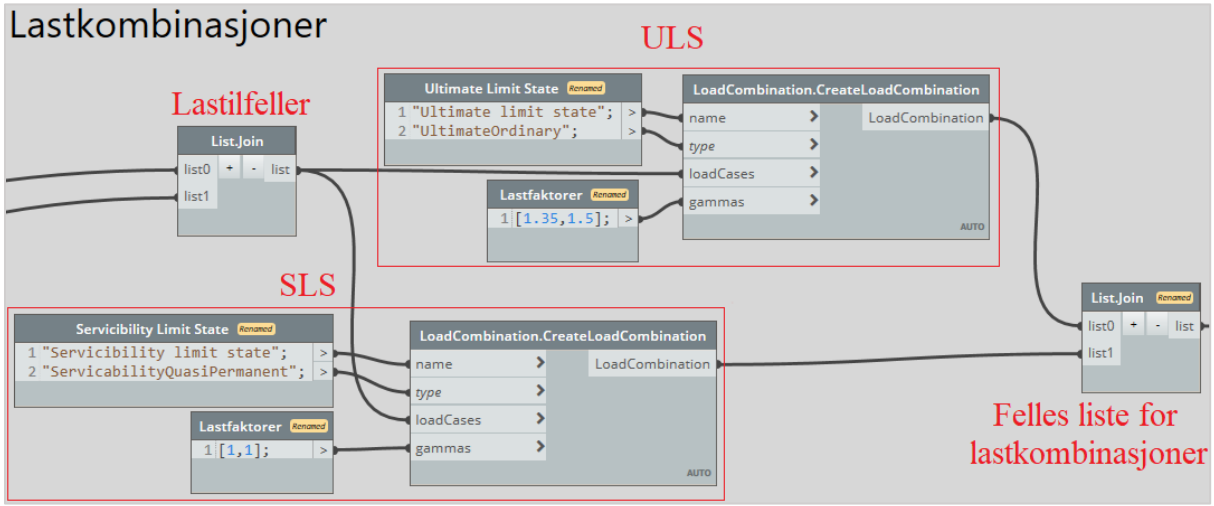
Laster og hvordan de påvirker en konstruksjon er veldig variabelt fra prosjekt til prosjekt, noe som gjør det vanskelig å automatisere. FEM-Design sin metode å lage og plassere ut laster er oversiktlig og lite tidkrevende. Å lage et script for laster er derfor ikke alltid tidsbesparende. I denne oppgaven er det kun egenvekt og en jevnt fordelt last på 5 kN/m^2 som påvirker konstruksjonen. Å automatisere enkle laster er gunstig, og dermed lages det et metode som tar for seg denne prosessen.

Først etableres lasttilfeller for hver last med noden ***LoadCase.CreateLoadCase***. Her defineres navn, *type¸og duration class* for hver last. Disse kombineres i en liste ved hjelp av ***List.Join*** for å lage en felles liste for lasttilfeller. Egenvekt defineres som en «DeadLoad» i Fem-Design. Dette gjør at FEM-Design automatisk legger inn en last basert på egenvekt til elementer i modellen. For tilleggslasten må det legges til en jevnt fordelt last på overflaten, ved bruk av ***SurfaceLoad.Uniform***. Verdien av lasten blir definert med ***Vector.ByCoodinates*** som lager en vektor i kraftretningen. Prosessen er vist i *figur 4.13*.



Figur 4.13: Viser hvordan lastilfeller og overflate laster blir laget ved bruk av Dynamo

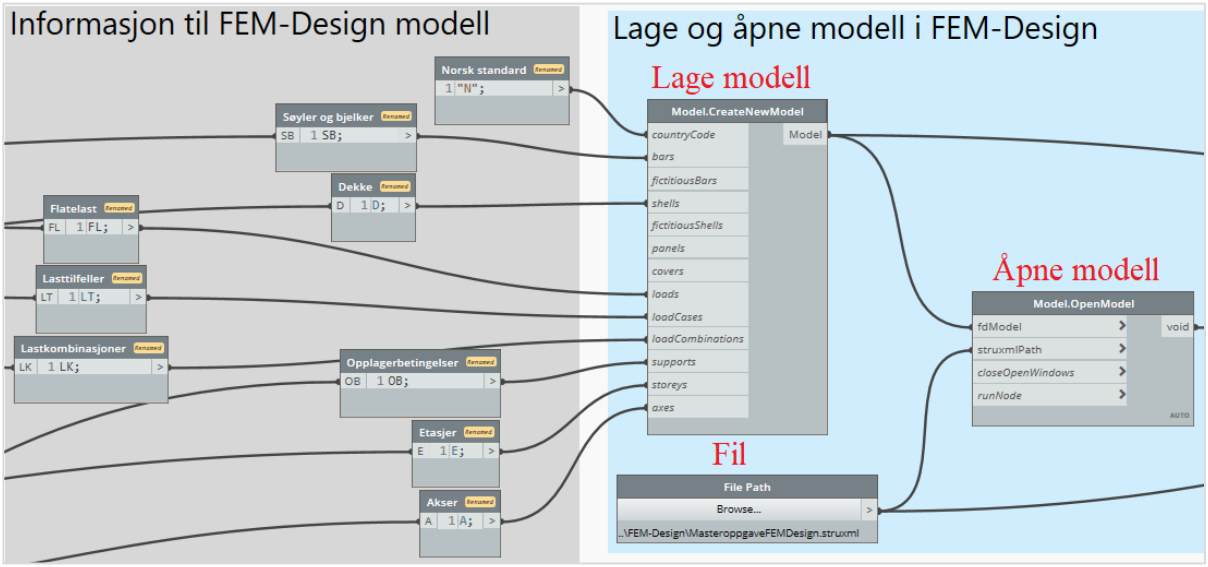
Til slutt etableres Lastkombinasjonene med ***LoadCombination.CreateLoadCombination*** basert på lastilfeller med tilhørende lastfaktorer, vist i *figur 4.14*. Dette gjøres for både *Ultimate Limite State* og *Servicibility Limit State*, som til slutt inkluderes i en felles liste med ***List.Join***



Figur 4.14 Viser hvordan lastkombinasjoner lager ved bruk av Dynamo

4.8 Lage, åpne og kjøre modell i FEM-Design

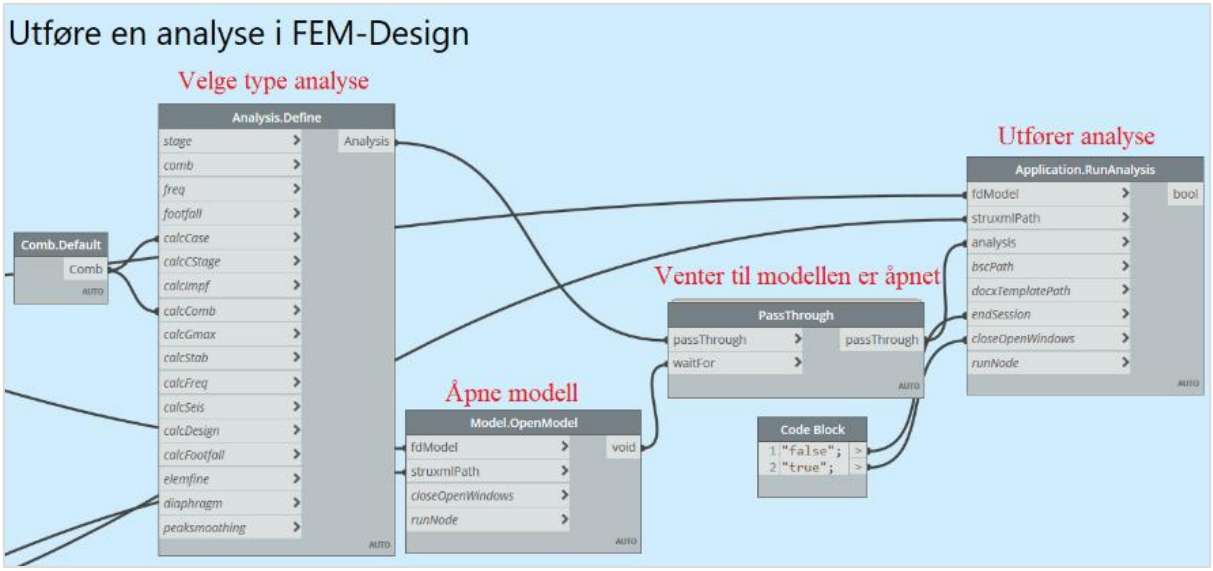
Relevant informasjon kobles på ***Model.CreateNewModel*** som lager en modell i FEM-Design. Videre gis modellen en fil med ***FilePath*** og åpnes med ***Model.OpenModel***, vist i *figur 4.15*



Figur 4.15: Lage og overføre modell til FEM-Design

Modellen er opprettet i FEM-Design og klar til å analyseres. ***Analysis.Define*** definerer hvilken type analyse som skal utføres før ***Application.RunAnalysis*** kjører analysen, vist i *figur 4.16*. Her må det oppgis informasjon om type analyse, modell og filnavn. Siden analysen er avhengig av en eksisterende modell, oppstår det en feilmelding når analysen kjører samtidig som overføringen. Derfor er det etablert en ***PassThrough*** node, som sier at modellen skal lages og åpnes før en analyse skal utføres.

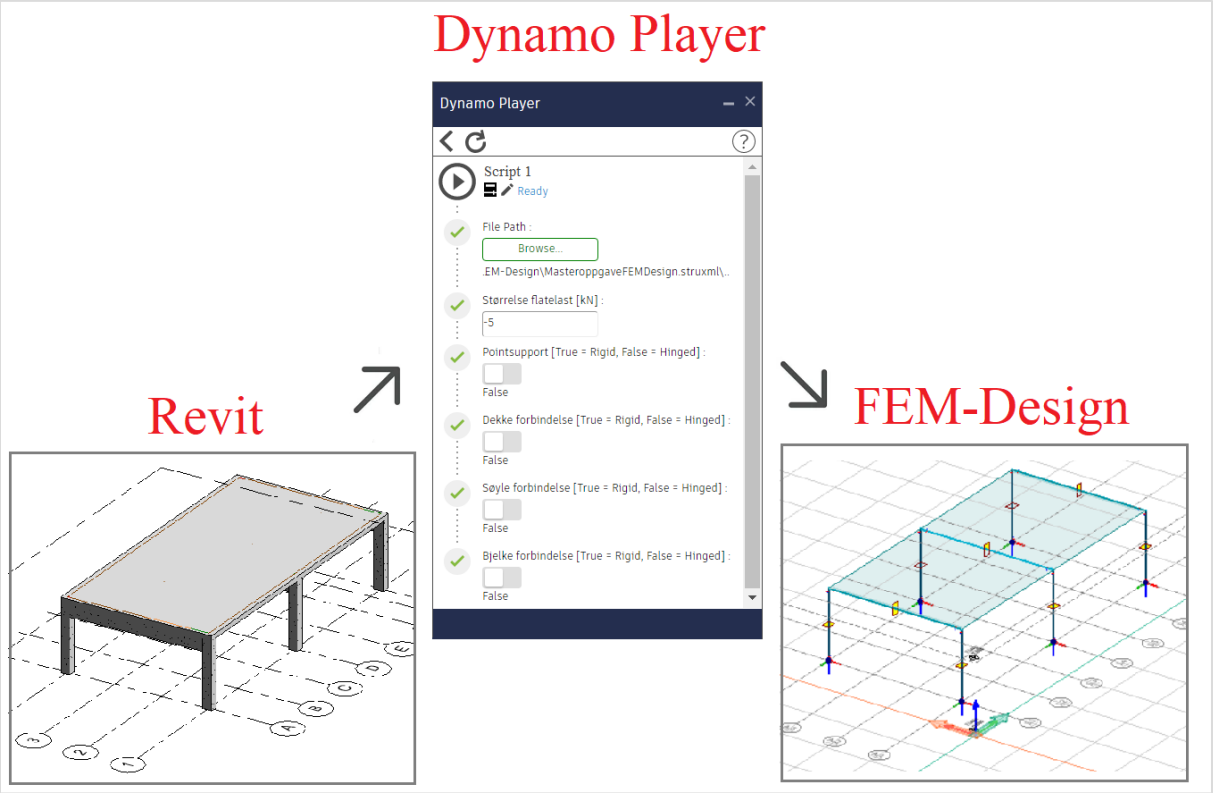
Application.RunAnalysis har som standardinnstilling at FEM-Design filen skal lukkes etter analyse og at et eksisterende vindu skal beholdes når scriptet kjøres på nytt. Dette er ikke ønskelig, derfor brukes *false* og *true* for å endre på innstillingene.



Figur 4.16: Viser hvordan en analyse kan velges og kjøres i FEM-Design ved bruk av Dynamo

4.9 Inputs for script 1

Ved bruk av Dynamo Player gjennomføres overføringen med kun 6 inputs som er mulig å endre på, vist i *figur 4.17*.



Figur 4.17: Dynamo Player kan brukes for å overføre en modell i Revit til FEM-Design

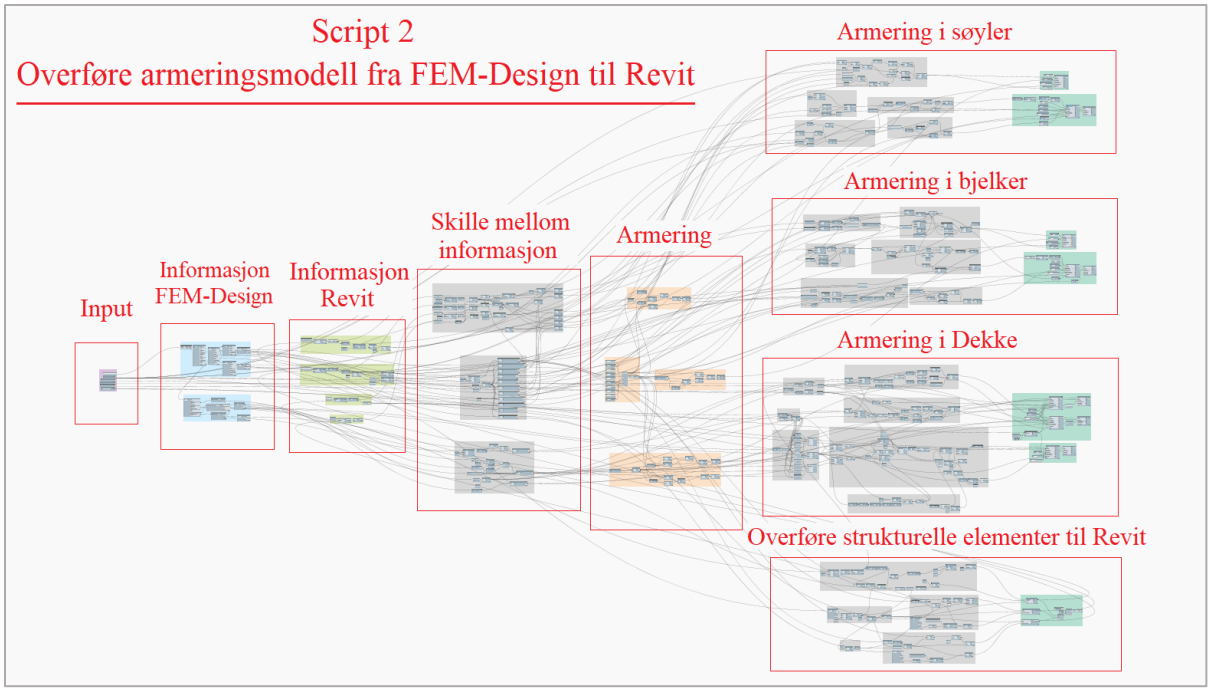
5 Forslag til script 2 - Overføre armeringsmodell i FEM-Design til Revit

Det bør ikke stoles blindt på at en automatisk overføring er helt korrekt. Derfor er det nødvendig at brukeren kontrollerer modellen, for å være sikker på at den er riktig. Deretter kan en armeringsanalyse utføres for å finne nødvendig armering til konstruksjonen. Basert på denne analysen lager FEM-Design en armeringsmodell som er utgangspunkt for dette scriptet.

Det er en tidkrevende prosess å lage en armeringsmodell i Revit. Siden det allerede eksisterer en armeringsmodell av konstruksjonen i FEM-Design, vil det være gunstig å lage et script som overfører denne til Revit. Dette vil føre til at brukeren slepper å bruke tid på å lage en armeringsmodell i både FEM-Design og Revit. Det spesifiseres at for *delkapittel 5.1* til og med *5.4*, overføres armeringsmodell tilbake til Revit modellen fra *kapittel 4*. Altså tar scriptet utgangspunkt i en eksisterende Revit modell.

5.1 Oversikt over script

Av samme grunn som nevnt i *delkapittel 4.1,* er det gunstig å ha en god organisering i scriptet som gjør at det enkelt kan finnes frem dersom behov for endringer. *Figur 5.1* gir en oversikt over scriptet og hvordan enkelte operasjoner er organisert. Fargekode er tilsvarende som for script 1, forutenom operasjoner relatert til armering som gis oransje farge.



Figur 5.1: Viser en oversikt over script 2

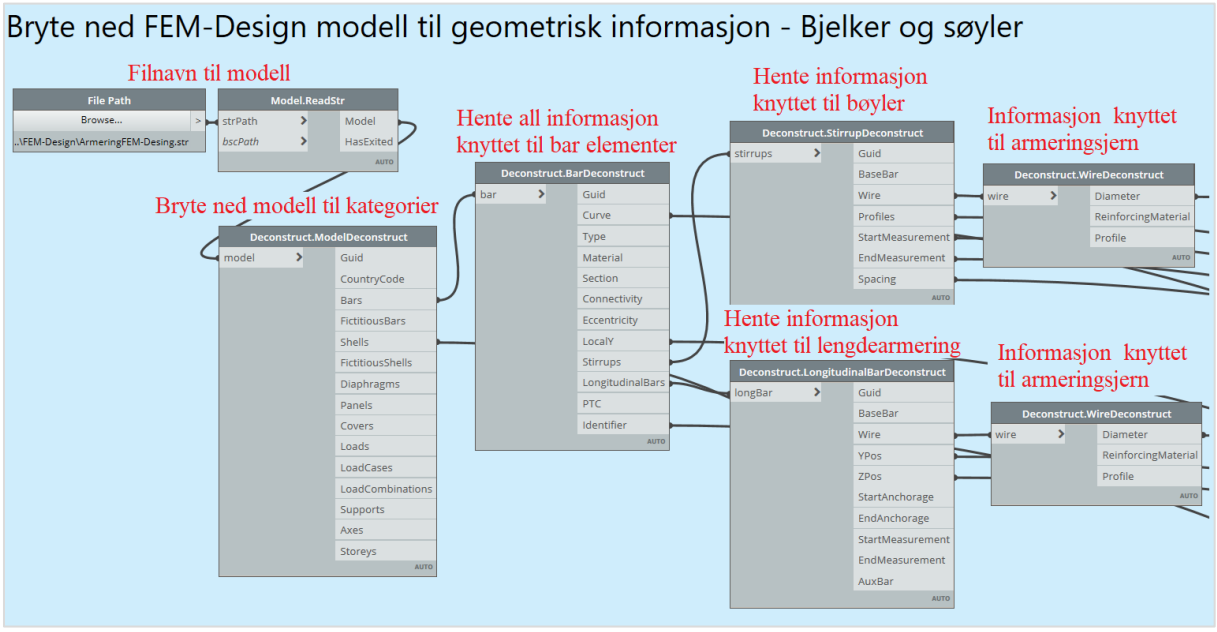
5.2 Overføring av armering – Søyler og bjelker

Dette delkapittelet vil ta for seg hvordan armering overføres fra FEM-Design til Revit. Siden søyler og bjelker er definert som et *bar element* i FEM-Design er fremgangsmetoden tilsvarende. For å unngå unødvendig repetisjon blir søyler og bjelker gjennomgått ilag og relevante forskjeller blir presentert.

5.2.1 Informasjon fra FEM-Design

For at Dynamo skal etablere en arbeidsflyt mellom Revit og FEM-Design brukes ***Model.ReadStr*** som henter informasjon fra en eksisterende FEM-modell. Denne informasjonen er utgangspunktet for å lage en modell i Revit, men det må gjøres noe korrigering.

Fra modellen benyttes ***Deconstruct.ModelDeconstruct*** for å hente alle kategorier. Siden søyler og bjelker er definert som *Bars* i FEM-Design brukes ***Deconstruct.BarDeconstruct*** for å hente relevant informasjon. Hvert *bar element* inneholder informasjon relatert til lengdearmering og bøyler som kan finnes med ***Deconstruct.LongitudinalBarDeconstruct*** og ***Deconstruct.StirrupDeconstruct***. Til slutt finnes all informasjon knyttet til hvert enkelt armeringsjern med ***Deconstruct.WireDeconstruct***. Denne prosessen er vist i *figur 5.2.*



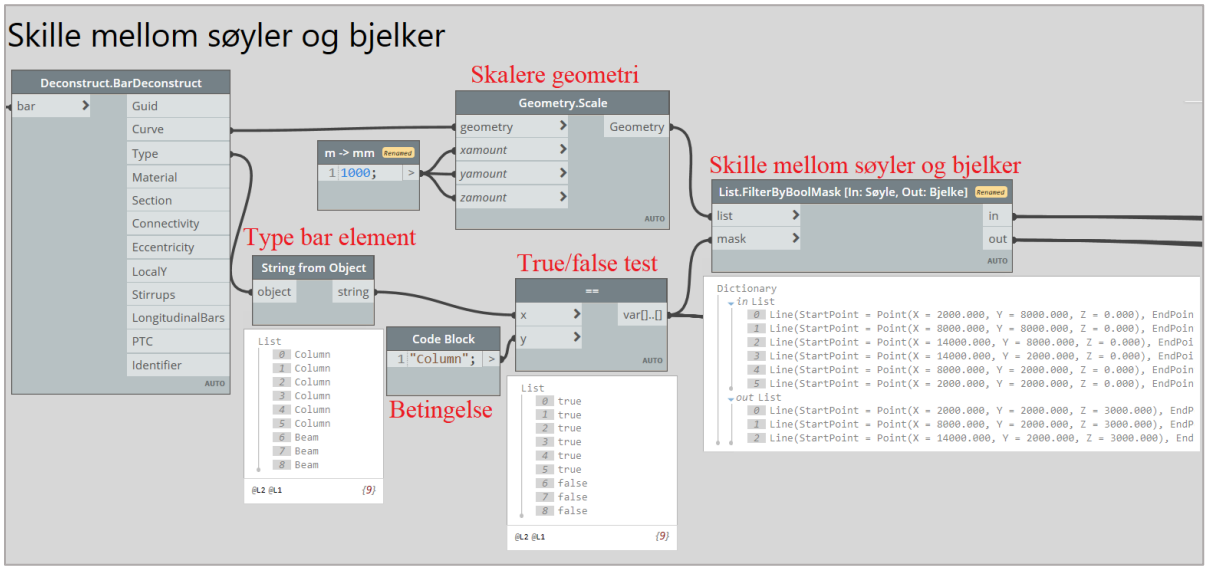
Figur 5.2: Hente ut informasjon relatert til søyler og bjelker fra en FEM-Design modell.

5.2.2 Lage et skille mellom bjelker og søyler

Ved overføring av armering er det viktig at den plasseres riktig og havner i riktig element. All informasjon om armering sin plassering er gitt utifra analytisk linje til et element. For at elementer blir tilført riktig armering, er det vesentlig å opprettholde en god listestruktur gjennom hele scriptet.

Siden søyler og bjelker er definert som et *bar element* i FEM-Design, må det lages en en metode som skiller mellom disse. ***Deconstruct.BarDeconstruct*** gir en liste over hvilken *Type* et element er og ***String from Object*** brukes å gjøre en *Type* om til tekst. Disse sammenlignes i en **==** node for å skape en «true/false» test.

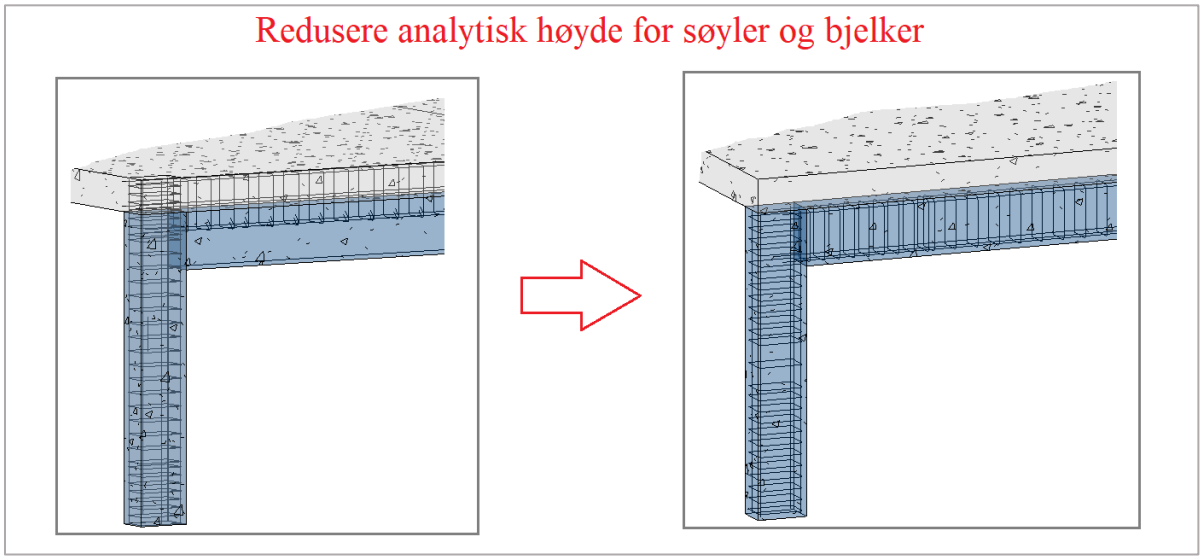
Dermed vil punkter i listen som er «Column» bli *true* og punkter med «Beam» resulterer i *false.* Denne testen brukes videre som betingelse i ***List.FilterByBoolMask***, som lager en egen liste for *true* verdier og en liste for *false* verdier. Altså, lages en liste for søyler og en liste for bjelker, illustrert i *figur 5.3.*



Figur 5.3: Skille mellom bjelker og søyler

5.2.3 Lage et skille mellom etasjer

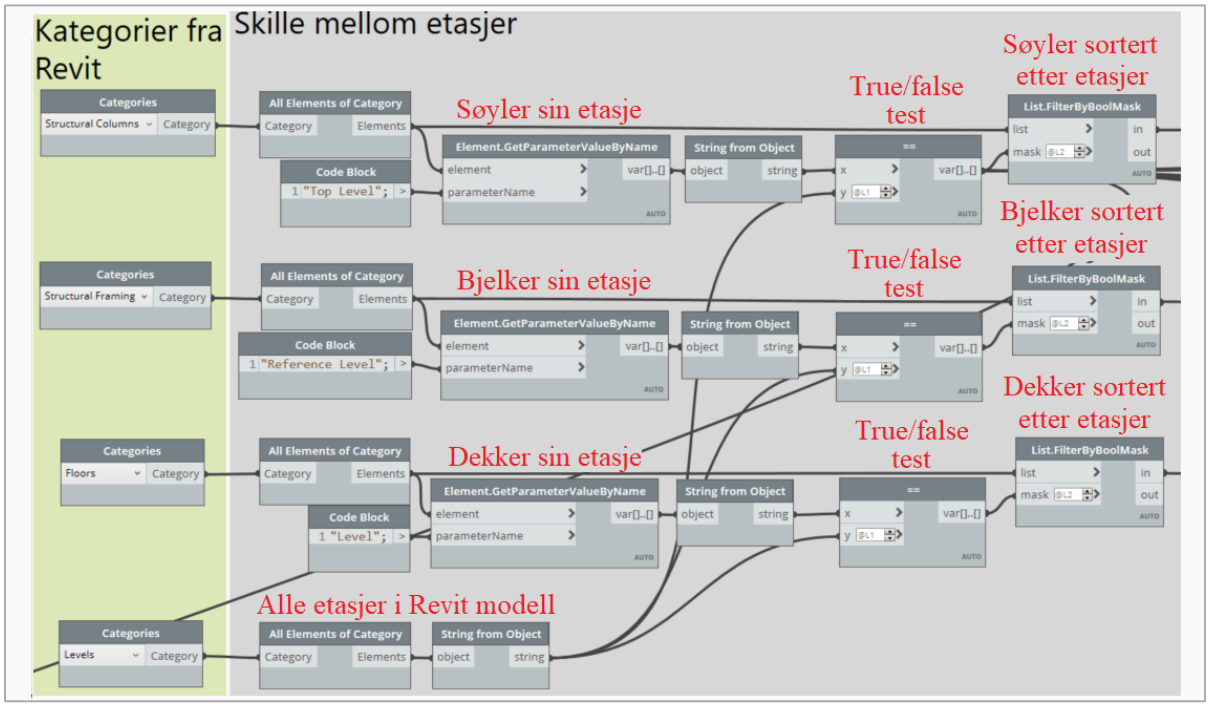
For en analytisk modell er toppen av søyler, bjelker og dekke i samme plan. Siden armering plasseres utifra analytiske linjer, vil armering for bjelker og søyler gå opp i dekke. Dermed er det nødvendig å redusere analytisk høyde for søyler og bjelker med overliggende dekketykkelse, vist i *figur 5.4*



Figur 5.4 Viser hvorfor analytisk høyde må reduseres for søyler og bjelker

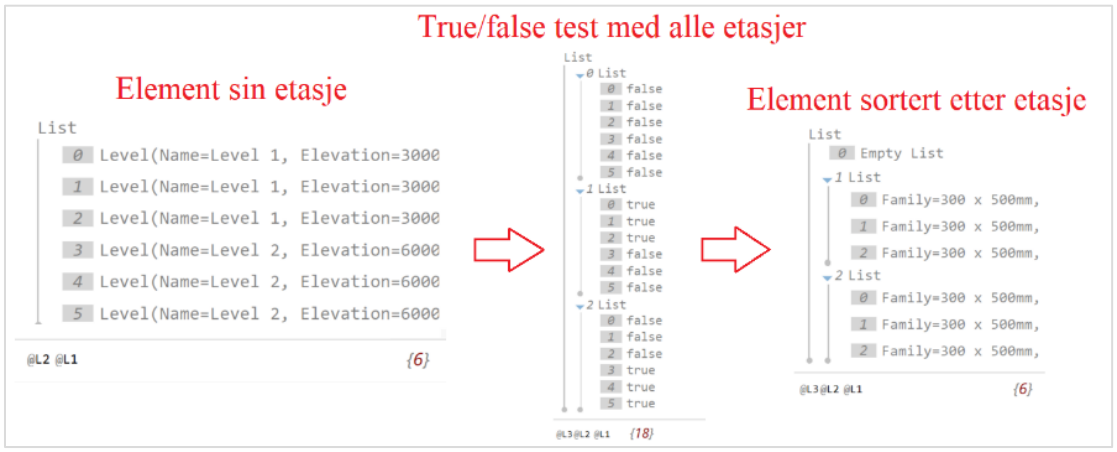
En modell kan bestå av flere etasjer med ulike dekketykkelser. For at riktig dekketykkelse brukes er det laget en metode som grupperer alle elementer i en etasje. Dermed vil alltid analytiske høyder for søyler og bjelker reduserest med overliggende dekketykkelse.

Denne metoden baserer seg på etasjer i Revit. *Levels* er en kategori i Revit som inneholder eksisterende etasjer i en Revit modell. Med kombinasjonen av ***Categories*** og ***All Elements of Categories*** henter Dynamo alle etasjer i Revit. Hvert element inneholder informasjon om sin tilhørende etasje, som kan finnes med ***Element.GetParameterValueByName***. Deretter er det ønskelig å etablere en liste som samler alle elementer i en etasje. Hvert element sammenlignes opp mot hver etasje med **==** for å lage en true/false test. Denne testen brukes videre som betingelse i ***ListFilterByBoolMask***, sammen med elementlisten. Resultatet er en liste over elementer som er sortert i etasjer, vist i *figur 5.5*.



Figur 5.5: Hvordan bjelker, søyler og dekker kan skilles i ulike etasjer

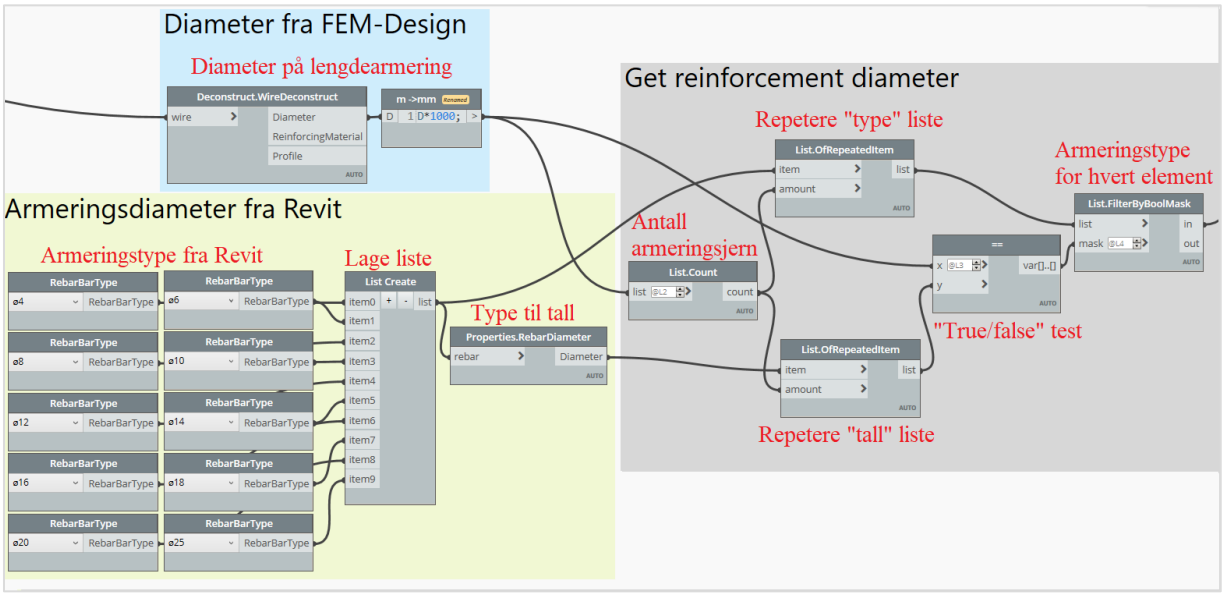
Denne metoden er også illustrert i listeform, som vist i *figur 5.6*



Figur 5.6: viser hvordan elementer kan skilles i ulike etasjer som listeform

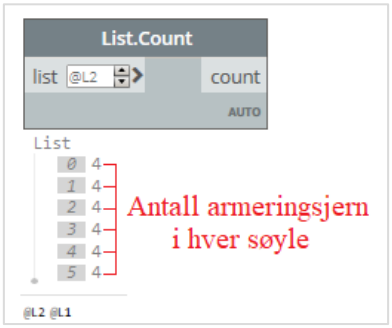
5.2.4 Lage armeringstype i Revit

For å overføre armering er det nødvendig å definere en armeringstype. Diameter på hvert ameringsjern hentes fra FEM-Design med ***Deconstruct.WireDeconstruct*** og blir oppgitt som en verdi. For å lage armering i Revit er det derimot krav om en armeringstype og ikke en verdi. Derfor må det lages en metode som lager en armeringstype basert på en diameter, vist i *figur 5.7*.

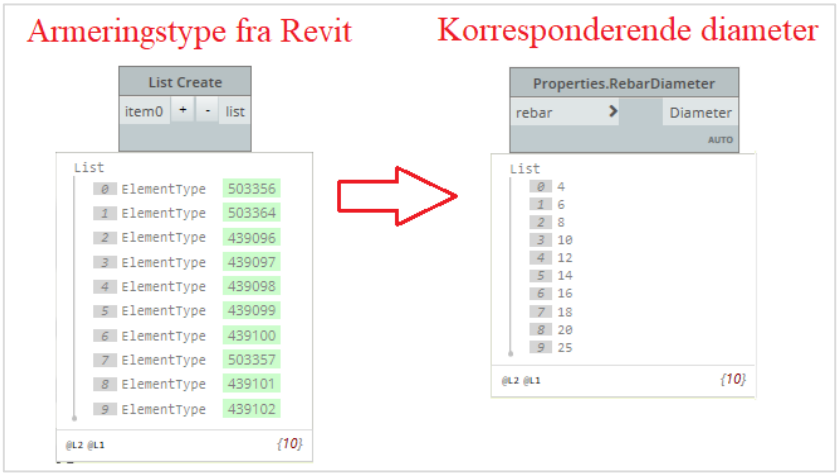


Figur 5.7: Hvordan en armeringstype lages i Revit basert på en diameter fra FEM-Design

For å finne antall armeringsjern fra FEM-Design brukes ***List.Count***, som gir en liste på 4 armeringsjern i hver av de 6 søylene, vist i *figur 5.8*. Videre brukes ***RebarBarType*** for å hente aktuelle armeringstyper fra Revit og samles i en liste med ***List.Create***, som skaper et «armeringsbibliotek». Armeringstypene sin korresponderene diameter finnes med ***Properties.RebarDiameter***, vist i *figur 5.9.*



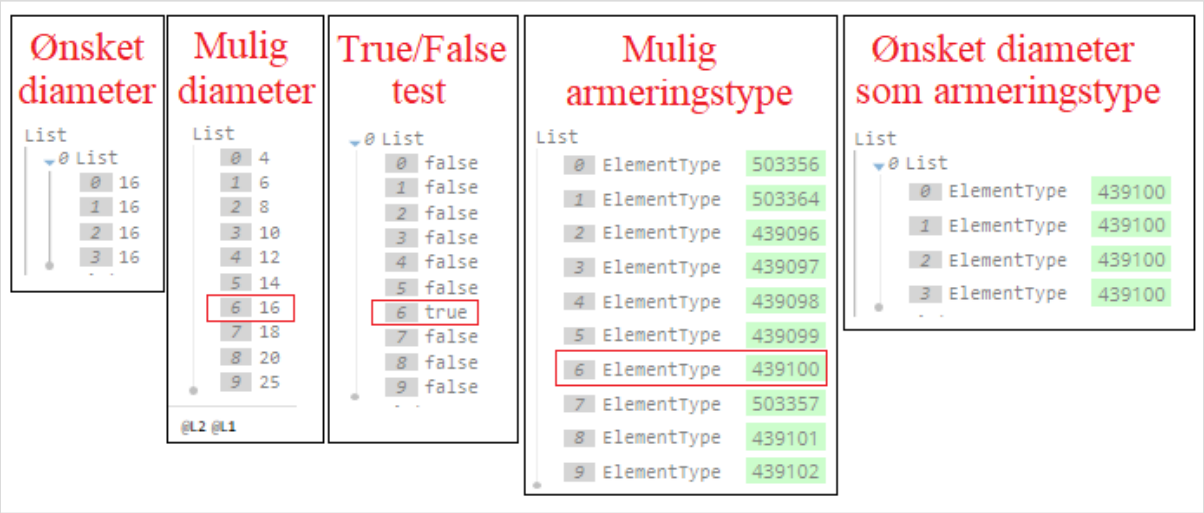
Figur 5.8: List count brukes for å telle hvor mange armeringsjern det er i hver søyle



Figur 5.9: Armeringstype fra Revit og korresponderende diameter

Hvert armeringsjern fra FEM-Design sjekkes opp mot armeringsbiblioteket. For å opprettholder listestruktur repeteres listen til armeringsbiblioteket like mange ganger som antall armeringsjern. Dette oppnås med ***List.OfRepeated*** Items der ***List.Count*** forteller hvor mange ganger listen skal repeteres.

Videre benyttes *==* for å lage en true/false test, der diameter fra FEM-Design sjekkes mot diameter listen fra Revit. Dette vil resultere i en liste med en *true* verdi og resten *false*. Denne listen brukes som en betingelse i ***List.FilterByBoolMask***, som henter armeringstype med korresponderende listenummer som *true*. Metoden er illustrert som listeform i *figur 5.10*



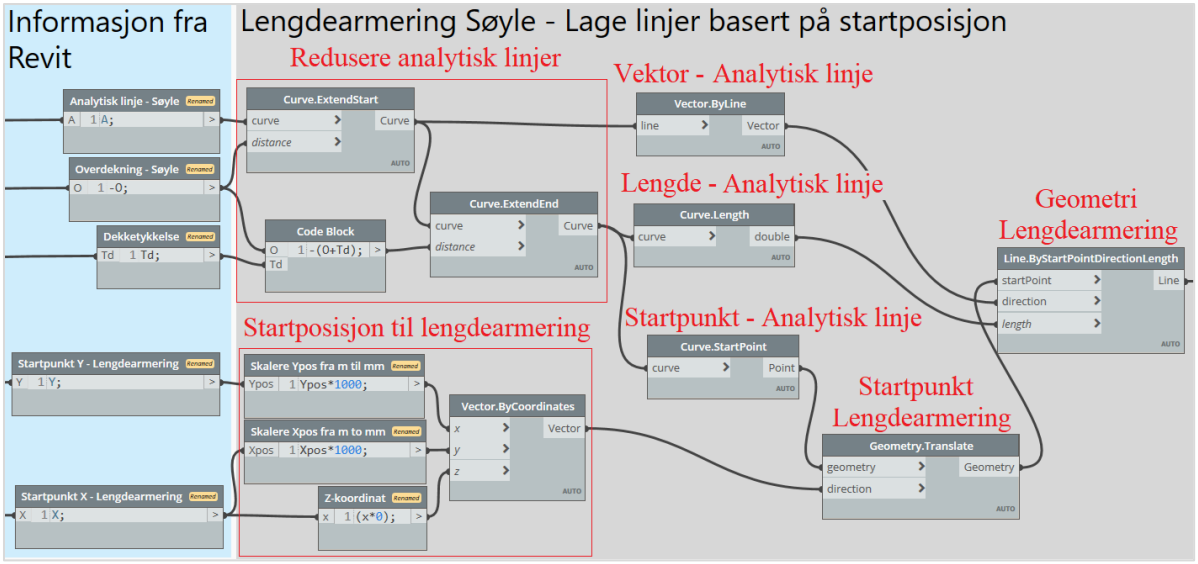
Figur 5.10: Armeringsdiameter for lengdearmering søyle

5.2.5 Lengdearmering i søyler

Lengdearmeringen tar utgangspunkt den analytiske linjen til en søyle, og dermed er det nødvendig å korrigere disse for å ta hensyn til overdekning. Dette oppnås med nodene ***Curve.ExtendStart*** og ***Curve.ExtendEnd*** som reduserer lengden i hver ende. Deretter finnes startpunktet til linjen med ***Curve.Startpoint*** som brukes som utgangspunkt for lengdearmering.

Posisjonen til lengdearmering er gitt fra FEM-Design som en avstand i X- og Y retning fra analytisk linje. Avstandene kobles til ***Vector.ByCordinates*** som lager en vektor som representerer avstand fra analytisk linje til lengdearmering. Vektoren og startpunktet for analytisk linje kobles videre på ***Geometry.Translate*** som flytter analytiske linje sitt startpunkt til startpunt for lengdearmering.

Siden lengdearmering er parallell med analytiske linje benyttes ***Vector.ByLine*** for å representere lengdearmeringen med en vektor. Videre lages geometrien med ***Line.ByStartPointDirectionLength***, der lengden hentes fra korrigerte linjer med ***Curve.Length***. Prosessen er vist i *figur 5.11.*

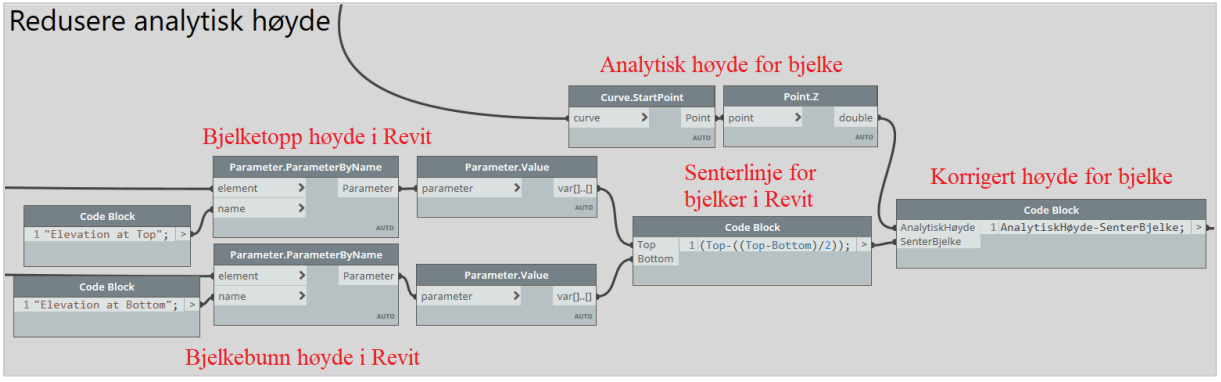


Figur 5.11: Viser hvordan geometri for lengdearmering til søyler kan lages basert på informasjon i FEM-Design

5.2.6 Lengdearmering i bjelker

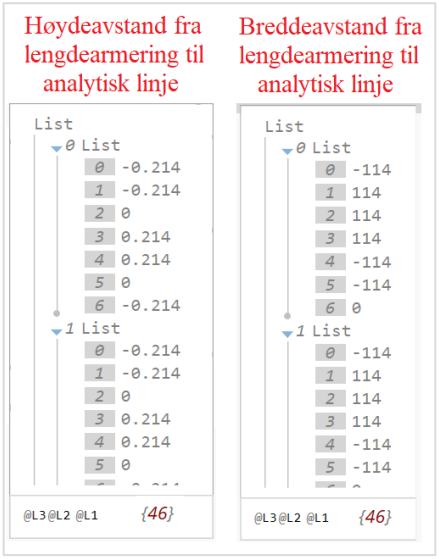
Fremgangsmåten for å overføre lengdearmering i bjelker har mye til felles med søyler, men det er noen ekstra utfordringer som må ta hensyn til. Tilsvarende som for en søyle er det ønskelig å redusere analytisk høyde, slik at armering ikke går opp i dekke.

For bjelker gjøres dette på en annen metode, siden informasjon om topp- og bunnhøyde av bjelken eksisterer i Revit. Disse verdiene hentes med ***Parameter.ParameterByName*** og ved hjelp av en enkel formel finnes høyden til senter av bjelke. Analytisk høyde for hver bjelke finnes basert på analytisk linje med kombinasjonen ***Curve.StartPoint*** og ***Point.Z***. Analytisk høyde reduserest med høyden til senter bjelke, for å finne avstand mellom analytisk høyde og senter bjelke, vist i *figur 5.12.*



Figur 5.12: Viser hvor mye analytisk høyde må reduseres med for å plasseres riktig i Revit

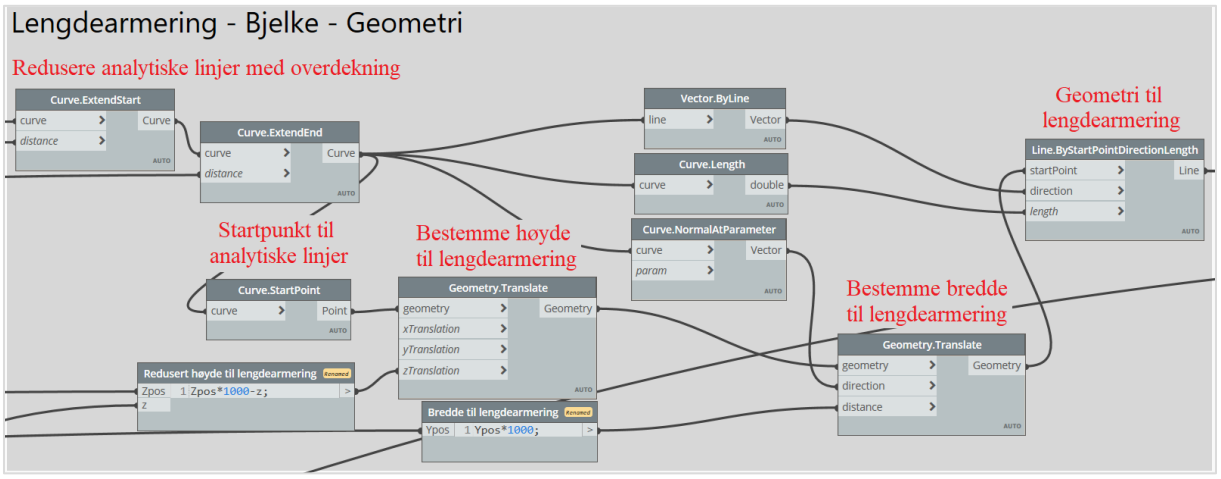
For å ta hensyn til horisontal overdekning i bjelker, reduseres analytiske linjer med ***Curve.ExtendStart*** *og* ***Curve.ExtendEnd***. Fra FEM-Design hentes informasjon om lengdearmering sin plassering i forhold til den analytiske linje. Her gis det en høydeavstand og en breddeavstand, vist i *figur 5.13*.



Figur 5.13 Viser avstand fra analytisk linje til lengdearmering

Videre finnes startpunkt til analytiske linjer med ***Curve.StartPoint***. Disse brukes i ***Geometry.Translate*** som lager nye punkt basert på en avstand. Først lages det punkter som bestemmer høyden til lengdearmering. Det må også trekkes fra differansen mellom analytisk høyde og senterlinje til bjelke.

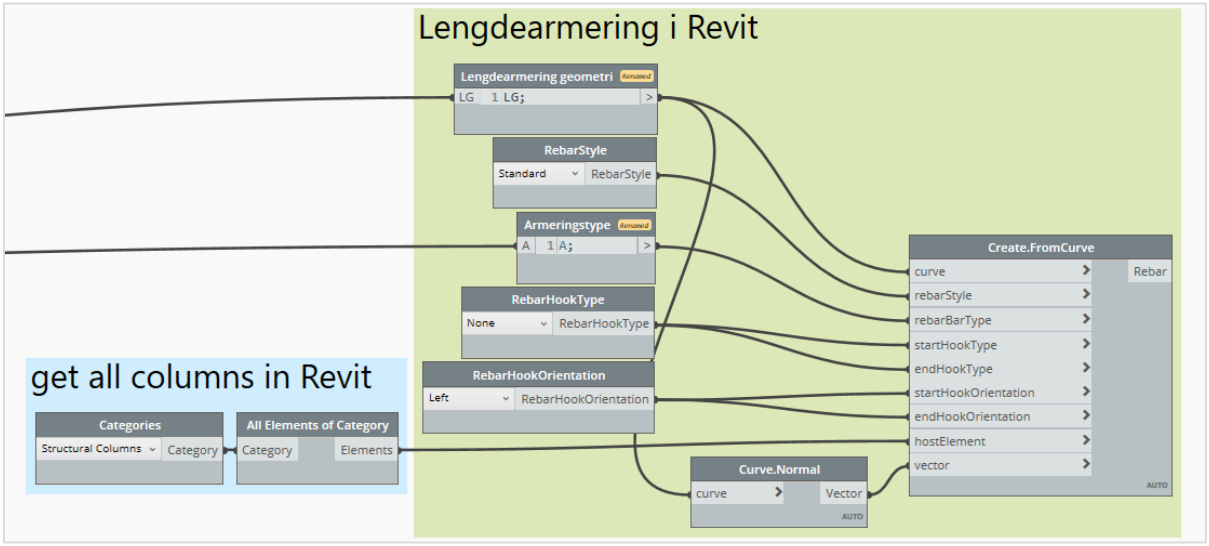
Bredde på en bjelke kan være i både X og Y retning, avhengig av hvilken retning bjelken går i. Dermed brukes ***Curve.NormalAtParameter*** som lager en vektor normalt på bjelken i plan. Basert på vektor og breddeavstand fra FEM-Design, brukes ***Geometry.Translate*** for å bestemme breddeposisjon til hver lengdearmering. Da er startpunktene til lengdearmeringen korrigert for både høyde og bredde. ***LineByStartPointDirectionLength*** benyttes for å lage en linje som representerer geometrien til lengdearmeringen. ***Curve.Length*** og ***Vector.ByLine*** henter lengde og retning fra analytiske linjer, som er redusert med overdekning. Prosessen er vist i *figur 5.14*



Figur 5.14: Korrigere analytiske linjer for å representere geometrien til lengdearmering i bjelker

5.2.7 Lage lengdearmering for søyler og bjelker i Revit

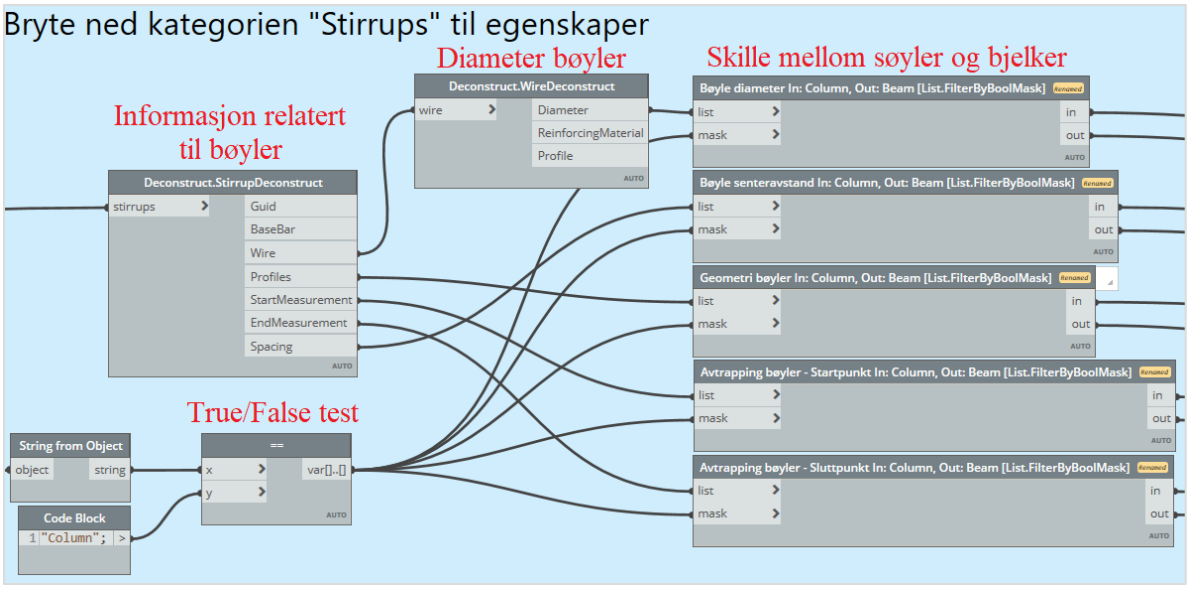
Før lengdearmeringen kan lages i Revit med ***Create.FromCurve***, krever den noe mer informasjon. ***Rebarstyle*** gir et valg mellom *Standard* og *Stirrup*, altså lengdearmering og bøylearmering, og velges som Standard. ***RebarHookType*** velges som *None* og ***RebarHookOrientation*** settes som *Left*. Videre må det definere et element som armeringen skal plasseres i. Dette gjøres med å velge kategori i ***Categories*** og deretter hente elementer med ***All Elements of Category***. Det må også defineres en normalvektor på lengdearmering som finnes med ***Curve.Normal*** fra geometrien til lengdearmeringen.



Figur 5.15: Lage lengdearmering i Revit

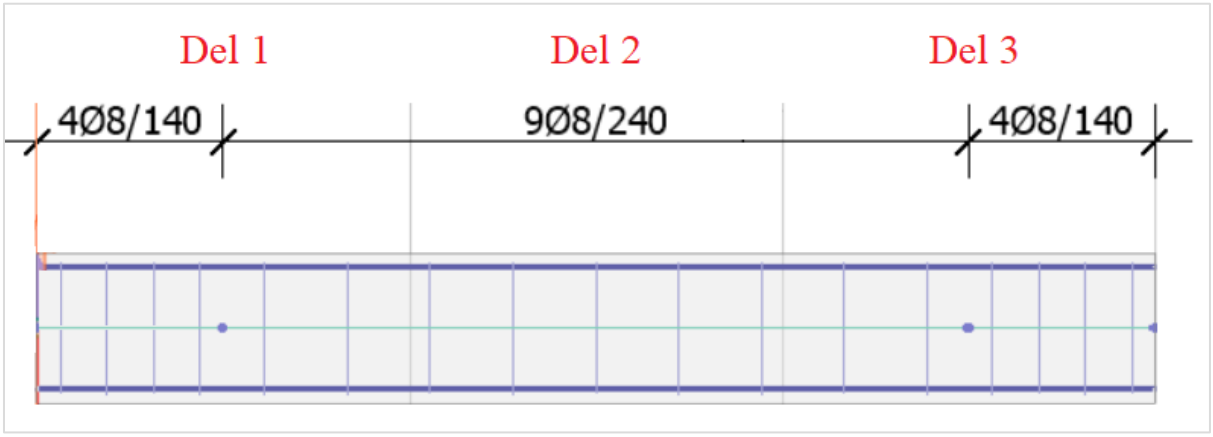
5.2.8 Bøylearmering i søyler

Informasjon relatert til bøylearmering hentes fra FEM-Design med ***Deconstruct.StirrupDeconstruct***. Her gis det informasjon om diameter, geometri, senteravstand, og avtrapping av senteravstand. Det må også skilles mellom søyler og bjelker på tilsvarende metode som i *delkapittel 5.2.2.*



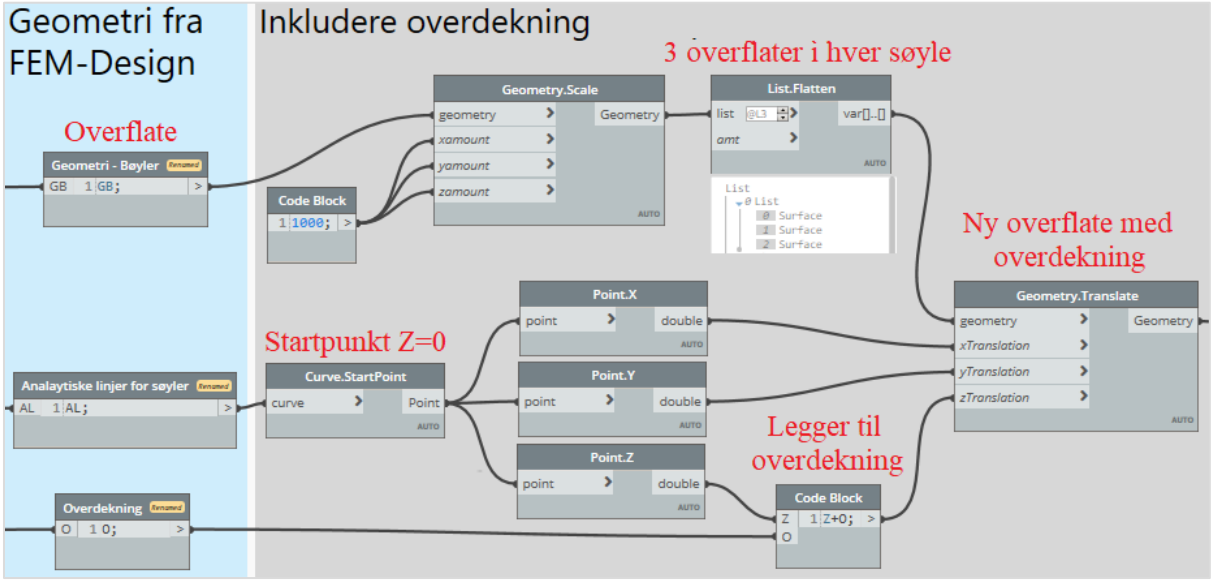
Figur 5.16: Bryte ned bøyler til egenskaper

Geometrien til en bøyle som hentes fra FEM-Design er gitt som en *Surface,* altså en overflate. Siden senteravstand mellom bøyler kan avtrappes, lager FEM-Design en gruppe for hver del som har lik senteravstand, vist i *figur 5.17*. Fra FEM-Design er senteravstand mellom hver bøyle redusert i bunn og topp av søyle, slik at det er totalt 3 deler. Hver av disse delene blir representert med en overflate fra FEM-Design, altså vil det overføres 3 overflater for hver søyle.



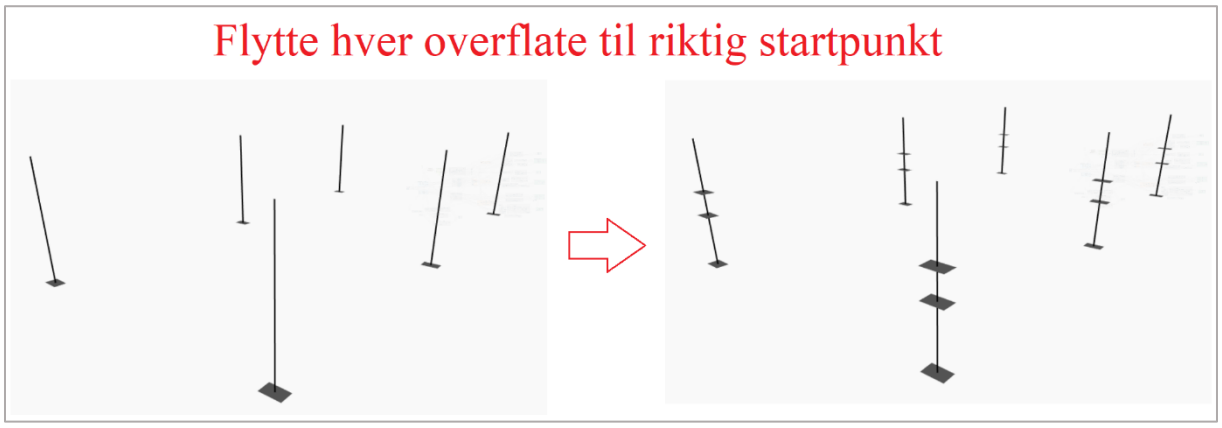
Figur 5.17: Avtrapping av senteravstand mellom bøyler

Disse overflatene inneholder ikke koordinater og må dermed gis koordinater, vist i *figur 5.18*. Som utgangspunkt benyttes startpunktet til søylen sin analytiske linje, som finnes med ***Curve.StartPoint***. ***Point.X, Point Y*** og ***PointZ*** benyttes for å hente koordinatene til punktet. Punktet er i bunnen av søylen, derfor legges overdekning til på Z koordinaten, før koordinatene kobles på ***Geometry.Translate***.



Figur 5.18: Viser hvordan overflater gis koordinater og tar hensyn til overdekning

Alle overflatene i en søyle blir flyttet til samme startkoordinater og må fordeles oppover i søylen, vist i *figur 5.19*. Dermed representerer overflatene starten på hver bøyledel. Startpunkt og sluttpunkt til hver overflate er hentet fra FEM-Design med ***Deconstruct.StirrupDeconstruct***.

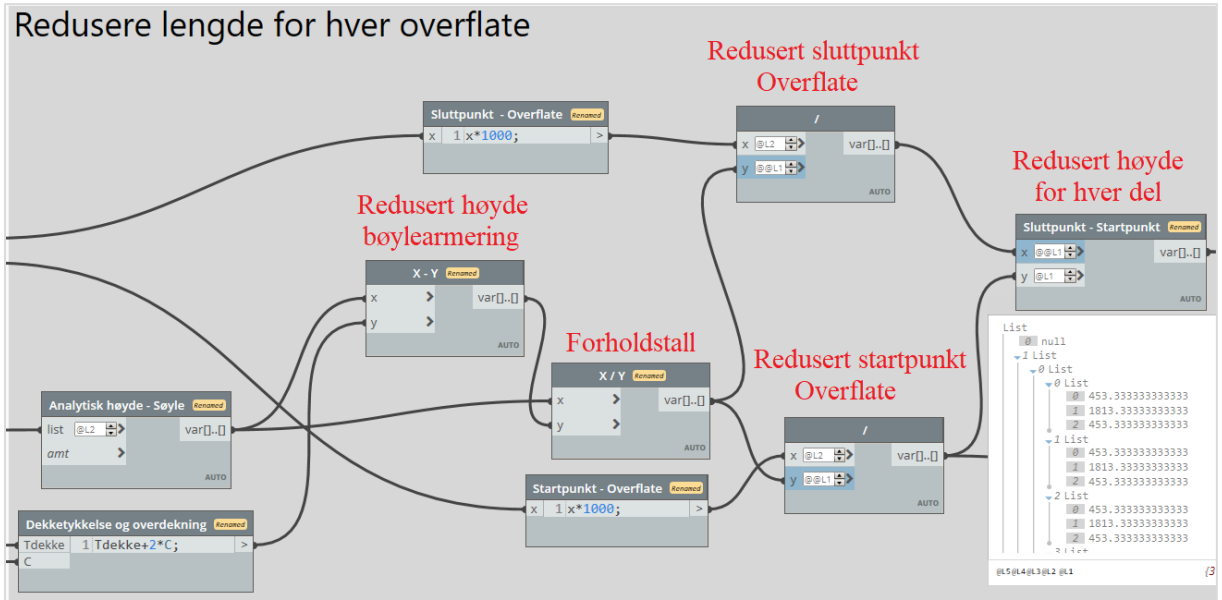


Figur 5.19 Flytte hver overflate til riktig startpunkt

FEM-Design sin armeringsmodell er laget utifra en analytisk modell, dermed vil bøylearmering gå opp til dekketopp. Derfor er det nødvendig å redusere høyden til bøylearmeringen med dekketykkelse. Denne reduserte avstanden fordeles utover på alle start- og sluttpunkt for overflatene, med å dele på et forholdstall. Dette forholdstallet er gitt med formelen:

$$\frac{Analytisk\ høyde\ for\ søyler}{Analytisk\ høyde - tykkelse\ dekke - overdekning}$$

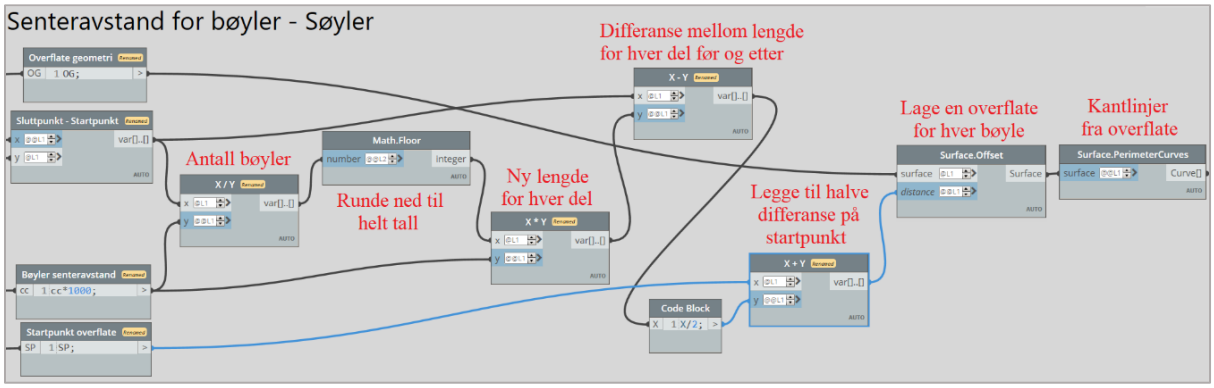
Figur 5.20 illustrerer hvordan lengden på hver overflate reduseres og en liste som viser hver del sin lengde.



Figur 5.20: Korrigere høyde for hver bøyle del

Neste steg er å definere lengden for hver bøyledel, som hentes fra FEM-Design. Her må det tas hensyn til at analytisk linje er redusert. Dette gjøres ved at hver bøyledel sin lengde deles med tilhørende senteravstand, noe som resulterer i antall bøyler i hver del. ***Math.Floor*** runder ned antallet til et helt tall. Det nye antallet deles på senteravstand for å finne en ny lengde for hver bøyledel. Siden antall bøyler er rundet ned, betyr det at den nye delen har en mindre lengde enn før. Derfor tas differansen mellom dem og deler på to, for å fordele den ekstra senteravstanden i bunn og topp.

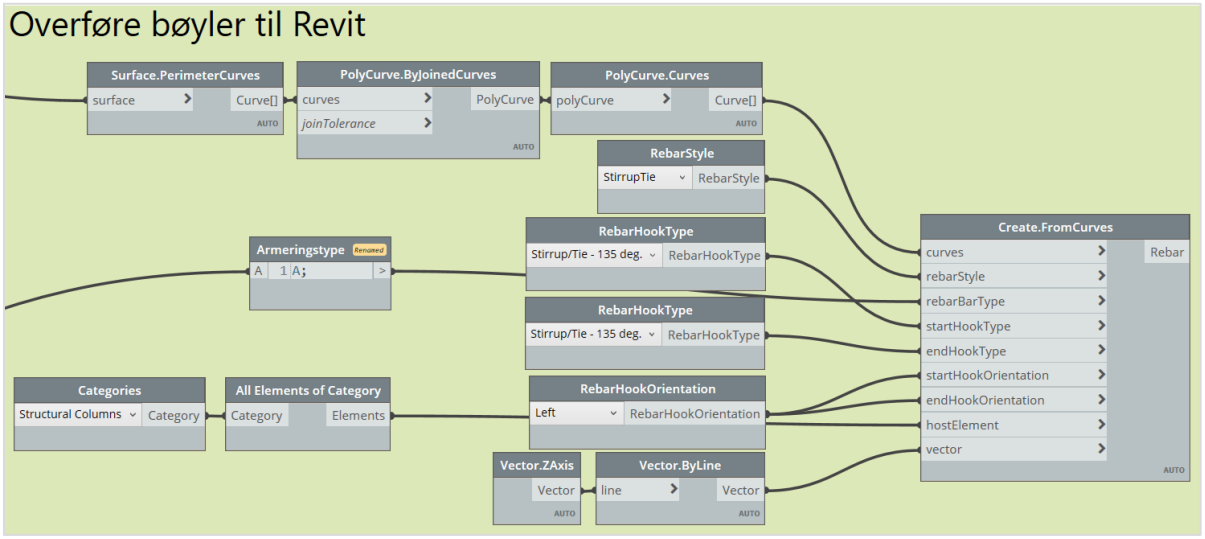
Senteravstanden brukes videre i ***Surface.Offset*** som lager en ny overflate på hver Z-koordinat som er gitt i listen. Da er hver bøyledel representert av en overflate som ligger med korrekte startkoordinater. For å lage en bøyle i Revit må geometrien være representert som kurver og ikke en overflate. Dette oppnås med ***Surface.PerimeterCurves*** som gir kantlinjene til en overflate. Prosessen er vist i *figur 5.21*



Figur 5.21: Plassere hver bøyledel på riktig startposisjon og hente kantlinjer fra en overflate

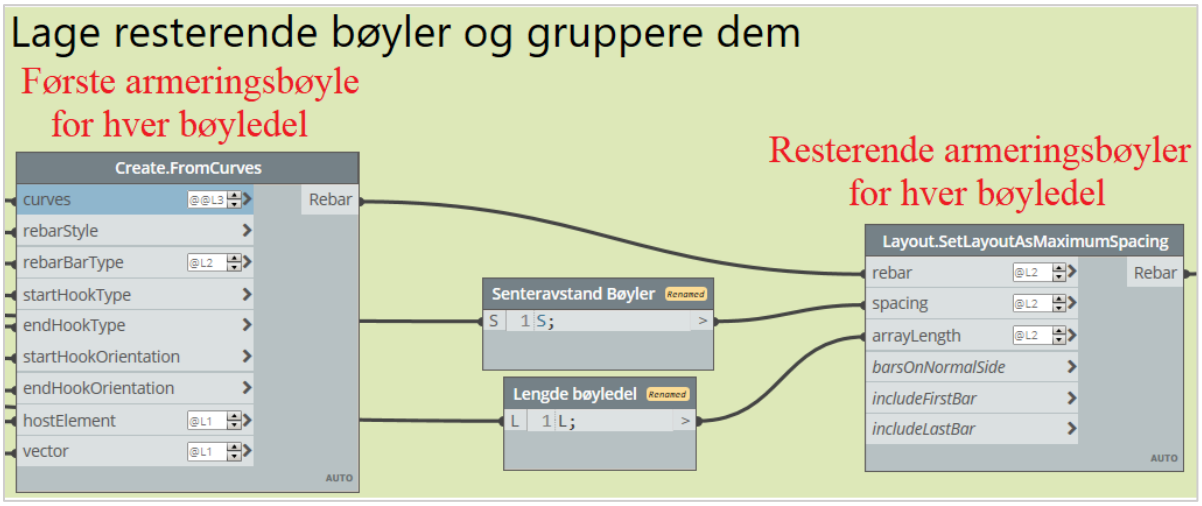
5.2.9 Lage armeringsbøyler i Revit

Kantlinjene til en overflate er definert som 4 uavhengige linjer, noe som gjør at Revit vil lage 4 seperate armeringsjern for hver bøyle. Dette unngås med ***PolyCurve.ByJoinedCurves*** som samler geometrien. ***Create.FromCurves*** vil ikke akseptere en *PolyCurve*, slik at ***PolyCurves.Curves*** må brukes for å gjøre en *PolyCurve* om til en *Curve.* Resten av prosessen er lik som overføring av lengdearmering som er gjennomgått i *delkapittel 5.2.7.*



Figur 5.22: Lage armeringsbøyler i Revit

Da er første armeringsjern laget for hver bøyledel. Videre lages resterende armeringsbøyler med ***Layout.SetLayoutAsMaximumSpacing*** som plasserer ut nye bøyler basert på senteravstanden mellom bøylene og lengden til hver bøyledel, vist i *figur 5.23.*

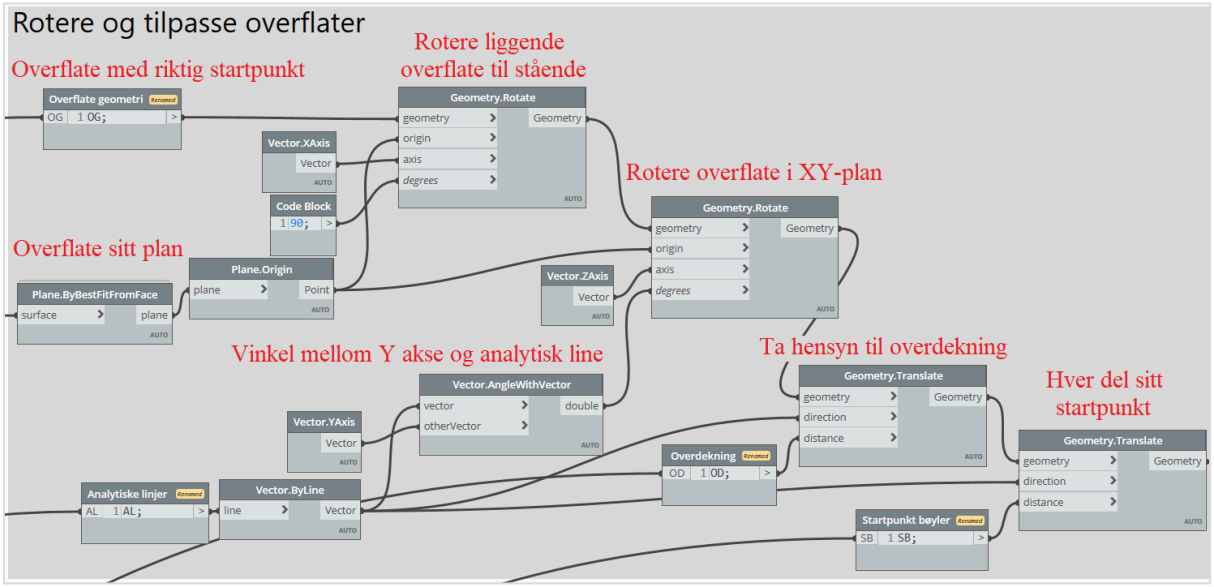


Figur 5.23: Lage resterende armeringsbøyler og gruppere hver bøyledel for seg selv

5.2.10 Bøylearmering i bjelker

På tilsvarende måte som for søyler, plasseres startpunkt for bøyler sin overflate ved start av analytisk linje. Siden FEM-Design ikke skiller mellom bjelker og søyler, er overflater fra FEM-Design overført som liggende geometri. Dermed er det nødvendig å rotere overflatene slik at bøylearmering ligger horisontalt. For å kunne rotere overflatene, defineres et punkt som overflatene skal rotere rundt. ***Plane.ByBestFitFromFace*** brukes for å lage et plan utifra en overflate og ***Plane.Origin*** gir midtpunktet som overflaten skal rotere rundt. ***Geometry.Rotate*** brukes for å rotere liggende overflater til stående.

Siden bjelker kan være i både X og Y retning er det også nødvendig at overflatene kan rotere slik at bøylearmering går i samme retning som bjelken. Her brukes ***Vector.AngleWithVector*** som gir vinkelen mellom bjelke og Y-aksen og forteller ***Geometry.Rotate*** hvor mange grader overflaten skal roteres i XY-planet. Denne prosessen er vist i *figur 5.24*

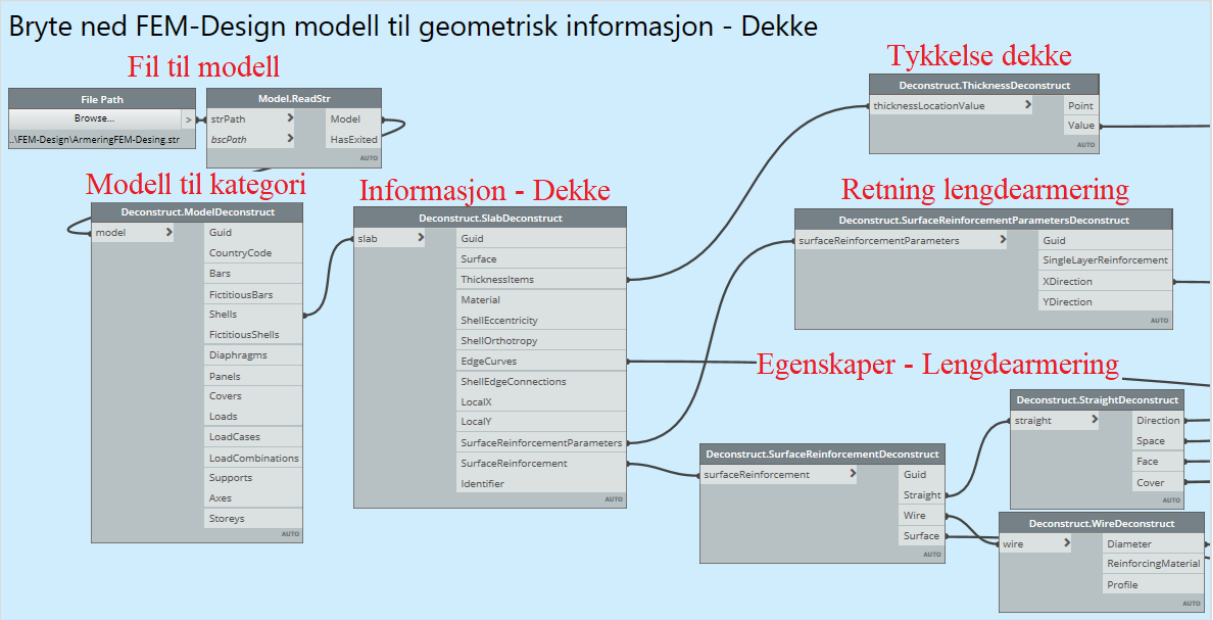


Figur 5.24: Rortere og tilpasse overflater slik at de plasseres riktig i bjelken

Da er retningen til overflatene definert og videre tas det hensyn til overdekning og at hver bøyledel har ulike startpunkt. Deretter blir bøylearmering overført til Revit på tilsvarende fremgangsmåte som vist i *delkapittel 5.2.9*

5.3 Overføring av armering - Dekke

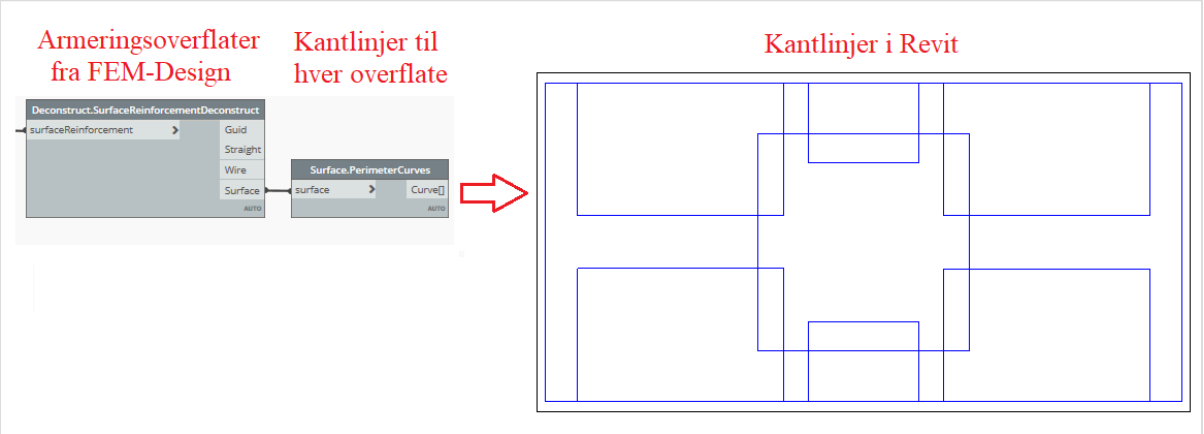
Siden et dekke er definert som et *shell element* i FEM-Design brukes ***Deconstruct.SlabDeconstruct*** for å hente informasjon relatert til dekke. ***Deconstruct.SurfaceReinforcemntDeconstruct*** brukes videre for å finne relevante egenskaper for dekke, som benyttes for å skape geometri til dekkearmeringen. Denne prosessen og egenskaper er illutstrert i *figur 5.25*.



Figur 5.25: Viser hvordan en modell kan brytest ned for å hente egenskaper til et dekke

5.3.1 Armeringssoner

Fra ***Deconstruct.SurfaceReinforcementDeconstruct*** representerer *Surface* outputen overflaten til hver armeringsone. Overflatene benyttes som utgangspunkt for å lage geometri til armering. Metoden innebærer å hente kantlinjer til overflater med ***Surface.PerimeterCurves***, vist i *figur 5.26*.

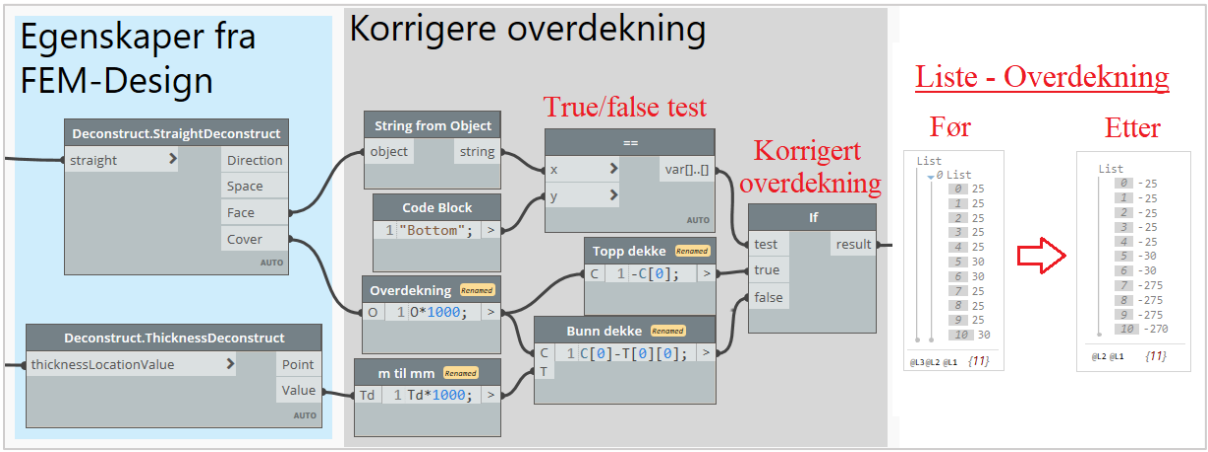


Figur 5.26: Viser hvordan armeringsoner kan gjøres om til kantlinjer

Armeringsoner inneholder kun informasjon om sine koordinater på den analytisk overflaten til dekke. For å hente armeringsegenskaper som overdekning, senteravstand og armeringretning benyttes ***DeconstructStraightDeconstruct***. Det er viktig at egenskapene gis til den tilhørende armeringsonen, slik at armering plasseres korrekt i dekke.

5.3.2 Overdekning i dekke

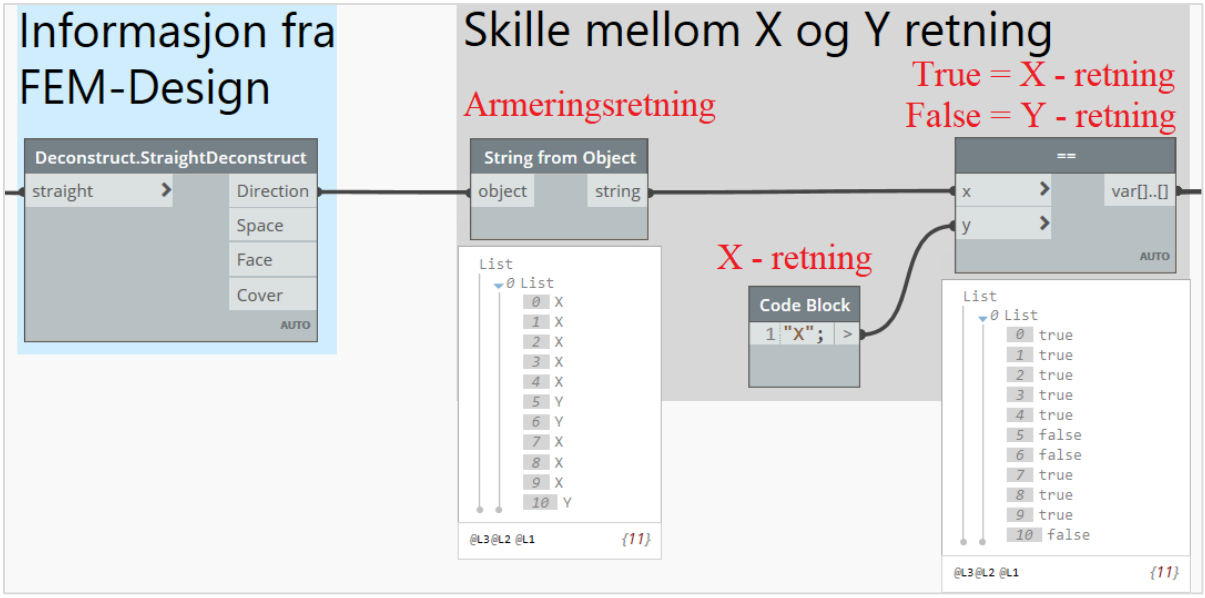
FEM-Design sine verdier på overdekning er målt fra enten topp eller bunn i dekke. En utfordring er at det ikke gis informasjon om hvor det er målt ifra. Dermed lages en metode for å skille mellom bunn og topp i dekke, slik at overdekningen blir riktig. Outputen *Face* gir informasjon om armeringen ligger i bunn eller topp av dekke. **==** brukes for å lage en true/false test som skiller mellom topp og bunn armering. Videre brukes testen i en ***If***-node, som kalkulerer overdekning fra toppen dersom *true* og overdekning fra bunnen dersom *false*, vist i *figur 5.27*.



Figur 5.27: Viser hvordan overdekning fra FEM-Design kan korrigeres slik at avstanden gis fra dekketopp

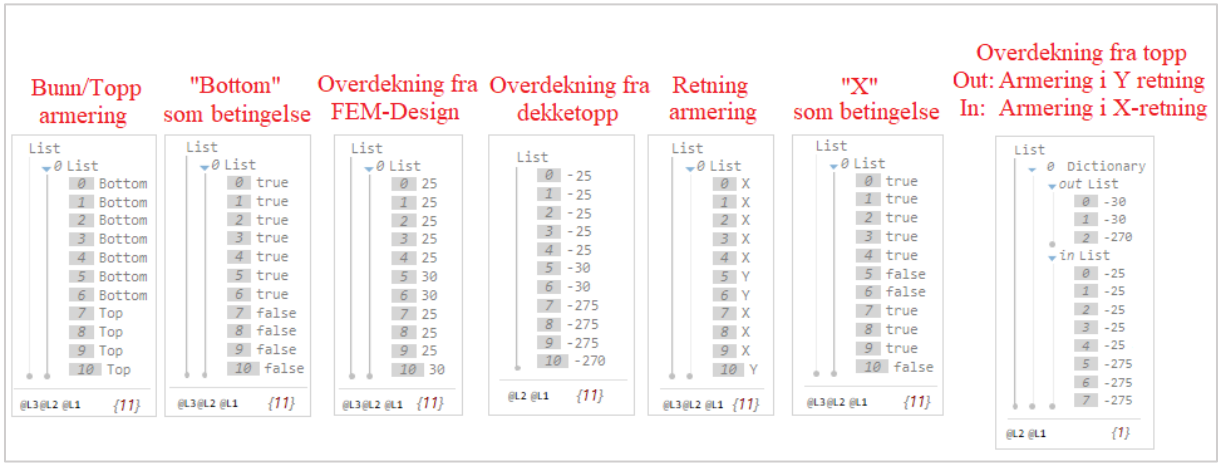
5.3.3 Lage et skille mellom armering i X- og Y retning

Det er nødvendig å skille armering i X og Y retning, vist i *figur 5.28*. Fra ***DeconstructStraightDeconstruct*** hentes outputen *Direction*, som gir en liste over armeringsretningen for hver armeringsone. Denne listen kan sjekkes opp mot betingelsen «X» i en **==** node, som fungerer som en true/false test. Det gjør at all armering i X retning blir kategorisert som *true*, mens armering i Y retning blir gitt som *false*.



Figur 5.28: Skille mellom X og Y retning

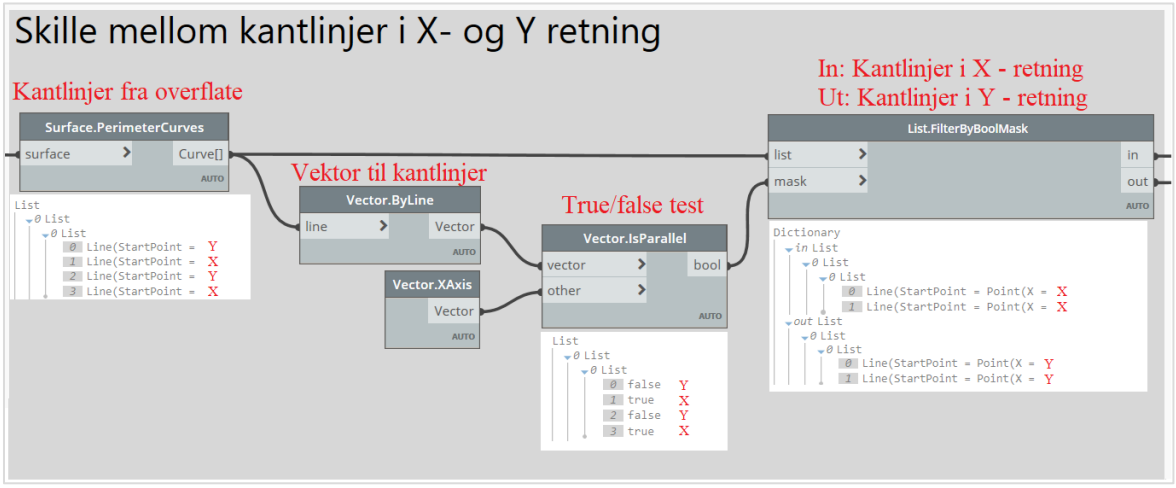
Denne listen er viktig videre i oppgaven for å lage geometri til lengdearmering, og blir videre omtalt som «test for armeringsretning». *Figur 5.29* viser i listeform hvordan overdekning blir korrigert og til slutt skillet mellom armeringsretning.



Figur 5.29: Viser i listeform hvordan overdekning korrigers og skillet mellom amreringsretning

5.3.4 Geometri for lengdearmering i X - retning

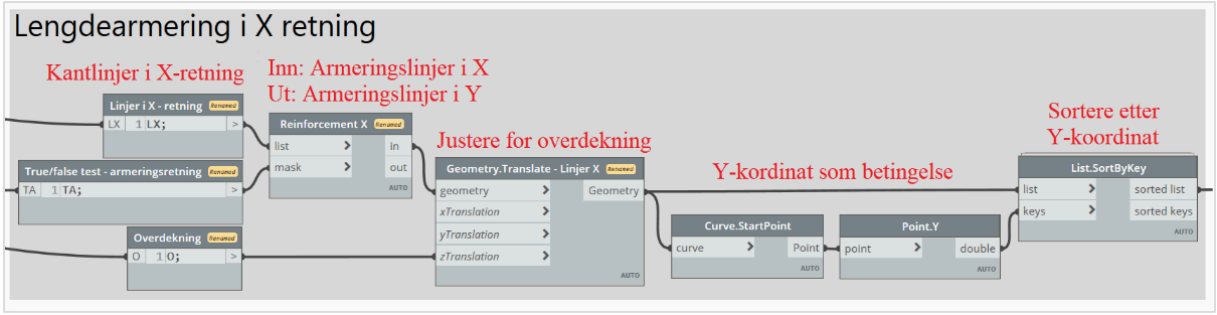
For å lage geometrien til lengdearmering er det tatt utgangspunkt i kantlinjer fra overflate til armeringsone, vist i *figur 5.26*. Kantlinjer representerer det første armeringsjernet og derfor må de skilles i X og Y retning. Dette gjøres ved å finne vektor til hver kantlinje med ***Vector.ByLine*** og bruke ***Vector.IsParallell*** for å sjekke om kantlinjen er parallell med X aksen. Siden overflaten er rektangulær geometri, vil alltid 2 kantlinjer gå i X aksen og 2 kantlinjer i Y aksen. Denne testen brukes som betingelse i ***List.FilterByBoolMask*** som lager en liste for kantlinjer i X retning og en liste for kantlinjer i Y retning.



Figur 5.30: Skille mellom kantlinjer i X- og Y retning

Armering i X retning representeres med kantlinjer i X retning. For soner der armeringsretning går i Y retning, benyttes ikke kantlinjer i X retning. Dermed må disse fjernes fra listen. Dette oppnås med å benytte test for armeringsretning, gitt i *figur 5.28,* som betingelse i ***List.FilterByBoolMask*** sammen med listen over kantlinjer. Da vil kantlinjer som representerer armering i X retning gis i en liste, mens kantlinjer i X retning som ikke brukes gis i en annen liste. Videre brukes ***Geometry.Translate*** for å justere kantlinjene til korrekt høyde i dekke.

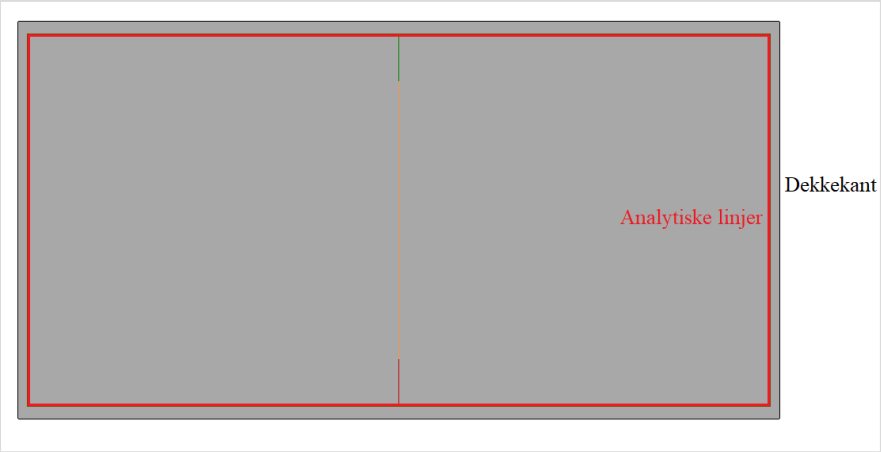
Det eksisterer 2 kantlinjer i X retning, men er kun behov for å bruke en av dem. Kombinasjonen ***Curve.StartPoint*** og ***Point.Y*** benyttes for å finne Y-koordinaten til de 2 kantlinjene. Deretter vil ***List.SortByKey*** sortere kantlinjer i X retning, utifra Y koordinaten sin. Noe som resulterer at kantlinje med lavest Y koordinat alltid brukes som første armeringsjern, vist i *figur 5.31.*



Figur 5.31: Viser hvordan geometrien for lengderetning i X-retning lages

5.3.5 Flytte armering fra analytisk linje til dekkekant

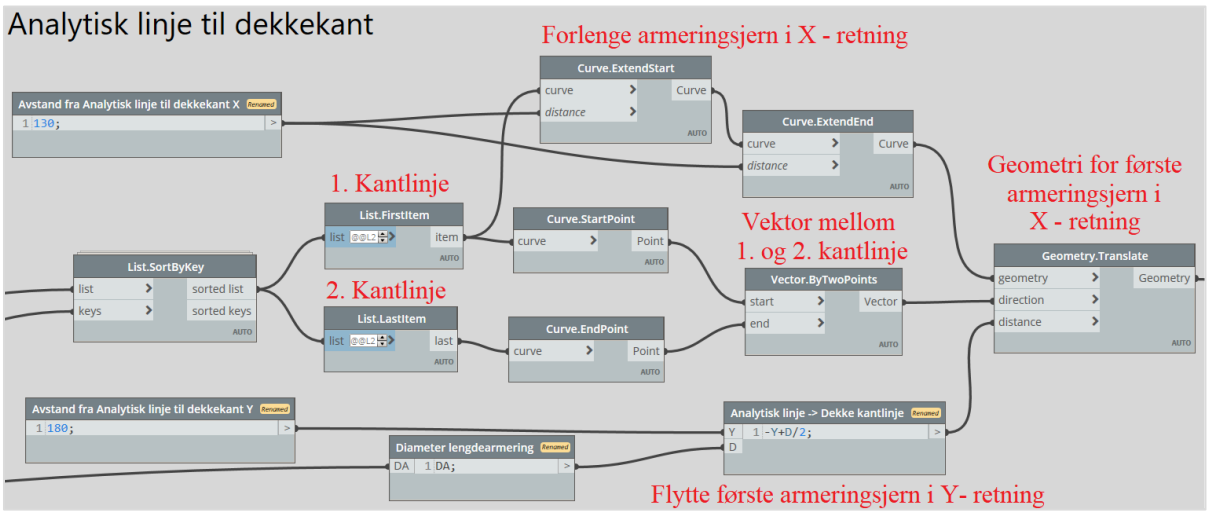
Siden armeringsonene fra FEM-Design tar utgangspunkt i analytiske linjer er det nødvendig å legge til den ekstra avstanden ut til dekkekant, vist i *figur 5.32*.



Figur 5.32: Viser analytiske linjer og dekkekanten til et dekke

For å ta hensyn til avstanden parallellt med armeringsjern, forlenges kantlinjer med ***Curve.ExtendStart*** *og* ***Curve.ExtendEnd***. Det er også nødvendig å legge til avtanden som ligger vinkelrett på armeringsjern. Dette gjøres med å bruke kantlinjene som er sortert etter Y-koordinat. ***List.FirstItem*** henter den første kantlinjen med lavest Y-koordinat, mens ***List.LastItem*** brukes for å hente den andre kantlinjen.

I Dynamo vil alltid vektor til kantlinjer i et rektangel følge klokken. For å lage en vektor mellom to parallelle kantlinjer, må vektor defineres med ***Vector.ByTwoPoints*** fra ene linjen sitt startpunkt og den andre linjen sitt sluttpunkt. Vektoren brukes videre i ***Geometry.Translate*** som flytter første armeringsjern til dekkekant. Prosessen er vist i *figur 5.33*

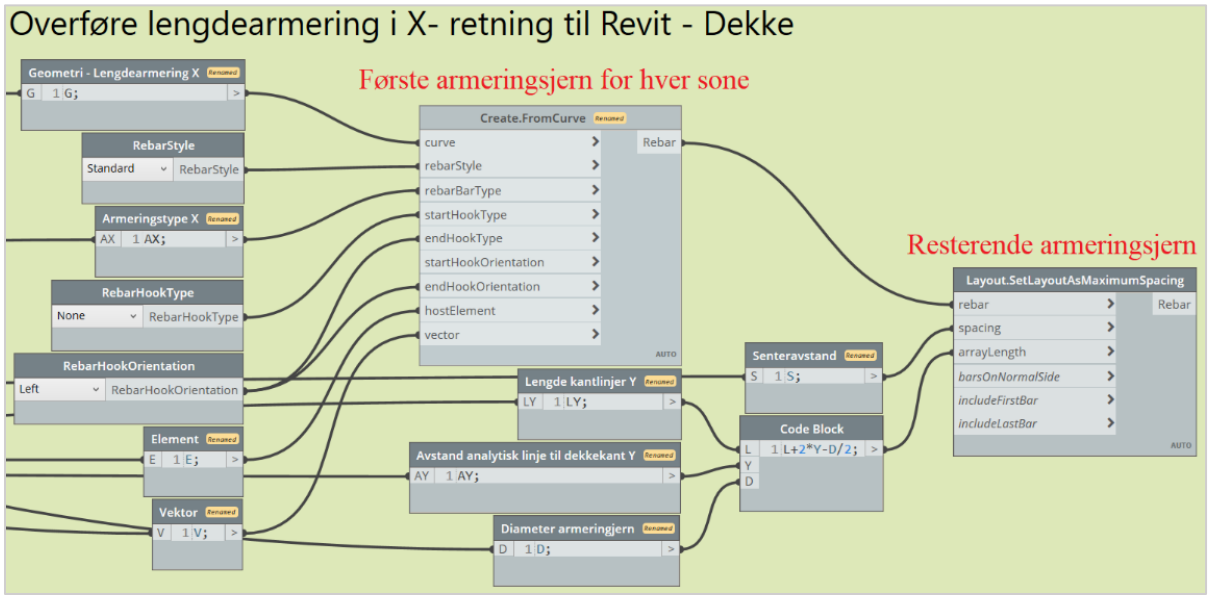


Figur 5.33: Viser hvordan analytiske linjer blir forskyvd ut til dekkekant

5.3.6 Lage lengdearmering for dekke i Revit

Da er geometrien for første armeringsjern etablert i X retning for hver sone. ***Create.FromCurve*** benyttes på tilsvarende måte som i *delkapittel 5.2.7*, og lager et armeringselement i Revit. Videre brukes ***Layout.SetLayoutAsMaximumSpacing*** for å lage de resterende armeringsjernene i en sone, samt gruppere armering i sonen. Her oppgis senteravstand og lengde på armeringsone fra FEM-Design, vist i *figur 5.34*

Tidligere i oppgaven er det etablert et skille mellom kantlinjer i X- og Y retning. Dermed kan ***Curve.Length*** brukes på kantlinjer i Y retning som forteller hvor lang en armeringsone er. Det må også legges til en lengde for avstand ut til dekkekant, samt halve diameter til armeringsjern. Denne prosessen må også repeteres for armering i Y retning.

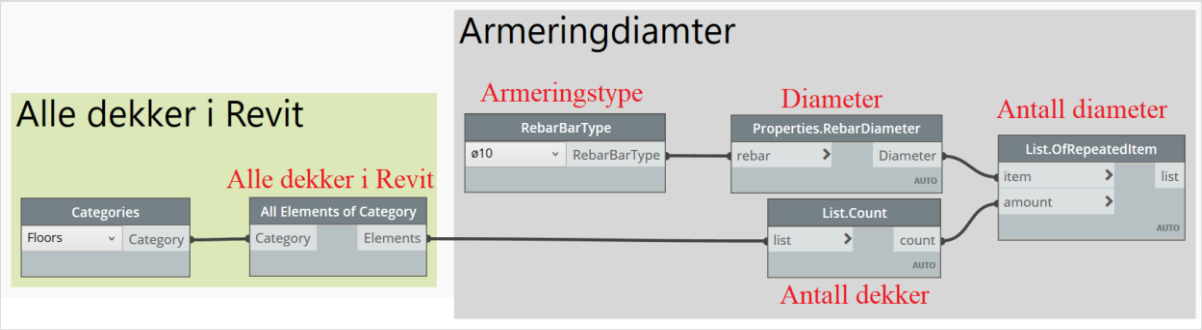


Figur 5.34: Lage lengdearmering i X-retning for et dekke

5.3.7 Endebøyler for dekke

Til nå i oppgaven er egenskaper til armering basert på FEM-Design sin armeringsmodell. Fra FEM-modellen eksisterer det derimot ingen informasjon relatert til endebøyler i et dekke. Derfor må geometri og egenskaper lages selv, noe som resulterer i noen inputs som kan endres på.

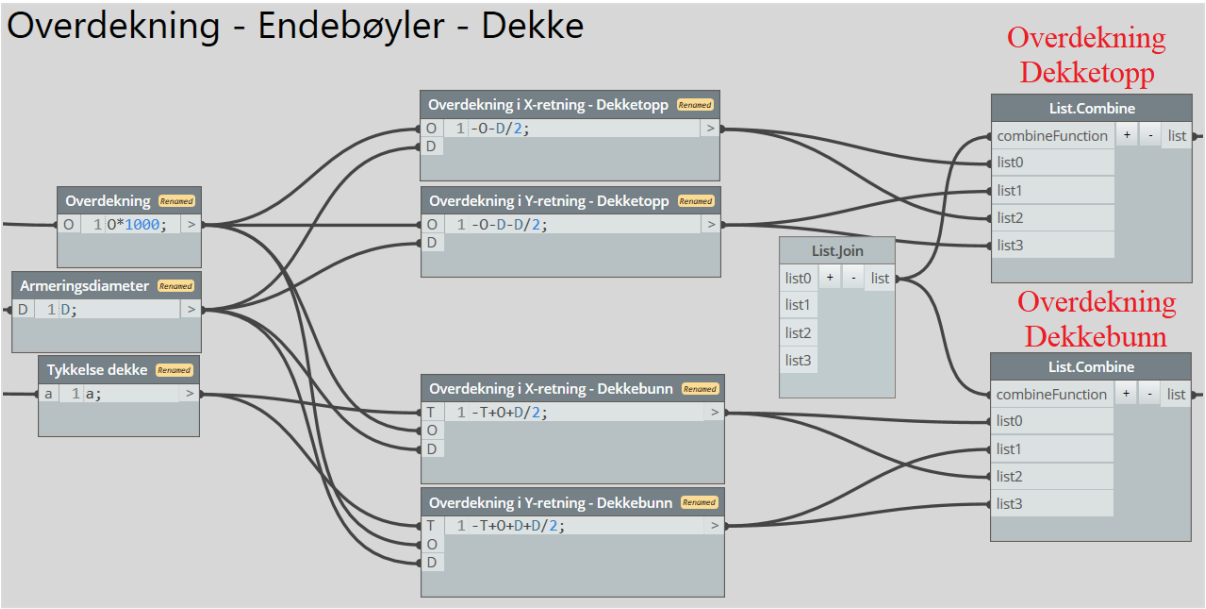
Først bestemmes armeringstype som skal brukes for endebøylene med bruk av ***RebarBarType***. ***Properties.RebarDiameter*** gir diameter til armeringstypen som repeterest i ***List.OfRepeatedItem*** med antall dekker som finnes med ***List.Count***. Dette gjøres for å opprettholde en listestruktur når det er flere enn et dekke i en modell. Prosessen er vist i *figur 5.35*



Figur 5.35: Tilpasse listestruktur for armeringsjern med listestruktur for gulv

5.3.8 Overdekning for endebøyler

Endebøyler ligger i det ytterste armeringslaget og for at de skal plasseres riktig brukes *Code Block* for å lage formler som gir riktig verdier for overdekning. For å unngå at endebøyler kolliderer, legges overdekning i X retning et lag høgere enn Y retning. Det er viktig at endebøyler ligger i samme armeringslag som lengdearmeringen i samme retning. Kombinasjonen av ***List.Combine*** og ***List.Join*** brukes for å etablere en liste over overdekning for hver gruppe med endebøyler, vist i *figur 5.36*

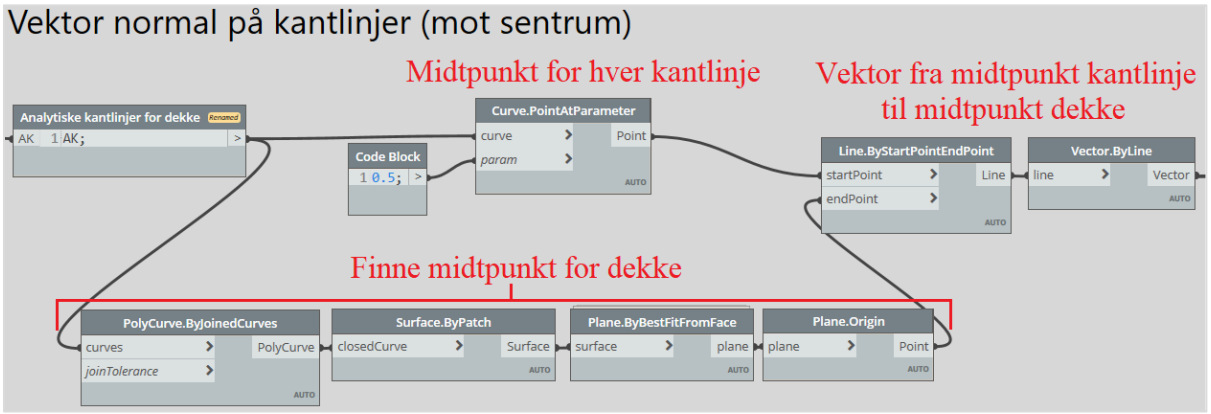


Figur 5.36: Lage formler som sørger for at overdekning alltid er tilpasset dekkekant

5.3.9 Vektor fra senter kantlinjer til senter dekke

Geometrien til endebøylene tar utgangspunkt i analytiske linjer til et dekke. For at endebøylene skal plasseres riktig, er det nødvendig å definere en vektor som alltid går fra midtpunkt kanlinje til senter av dekke. Først finnes midtpunktet til hver kantlinje med ***Curve.PointAtParameter,*** der 0,5 benyttes som parameter.

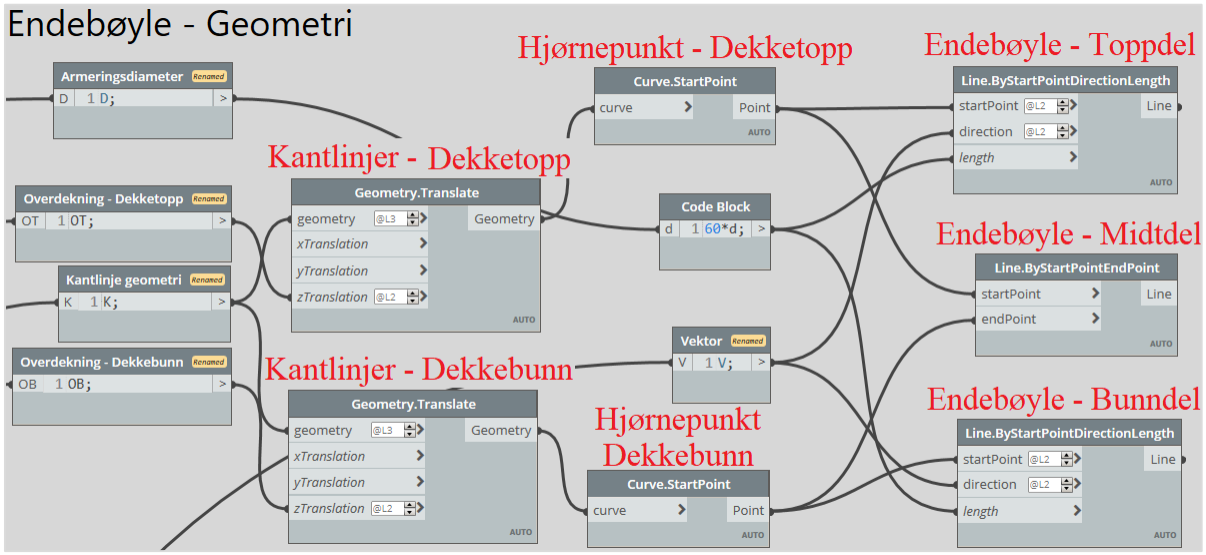
For å finne senter av dekke samles kantlinjene med ***PolyCurve.ByJoinedCurves*** for å lage en overflate med ***Surface.ByPatch***. ***Plane.ByBestFitFromFace*** gir et plan basert på en overflate og ***Plane.Origin*** finner midtpunktet til planet, altså midtpunktet til dekke. Til slutt lages en linje fra midtpunkt kanlinjer til senter dekke med ***Line.ByStartPointEndPoint***. Denne linjen brukes for å etablere en vektor med ***Vector.ByLine.*** Denne prosessen er vist i *figur 5.37*.



Figur 5.37 Viser hvordan en vektor lages mellom kantlinjer og senter dekke

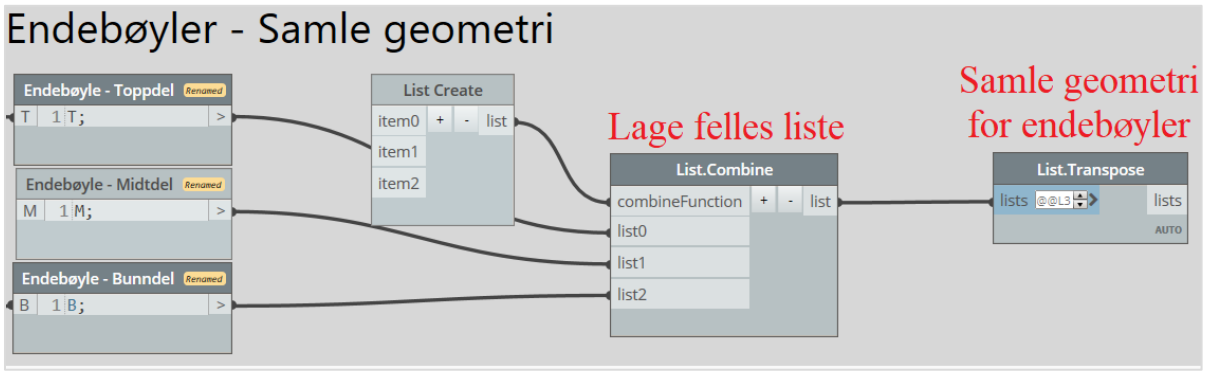
5.3.10 Lage geometri for endebøyler

Listen for overdekning målt fra dekketopp, definert i *delkapittel 5.3.6,* kobles på Z koordinat i ***Geometry.Translate*** for å flytte analytiske kantlinjer til dekketopp. Deretter finnes hjørnepunktene til kantlinjene med ***Curve.StartPoint***. Fra hjørnepunkter lager ***Line.ByStartPointDirectionLength*** en linje mot sentrum av dekke, med vektoren som er definert i *figur 5.36*. Her oppgis avstanden til linjen som representerer lengden til toppdelen av endebøylen, altså 60 ganger diameter til endebøyle. Samme prossess repeteres for overdekning målt fra dekkebunn. For å finne midtdel av endebøylen brukes ***Line.ByStartPointEndPoint*** som lager en linje mellom hjørnepunkt fra dekketopp til dekkebunn. Denne prosessen er vist i *figur 5.38.*



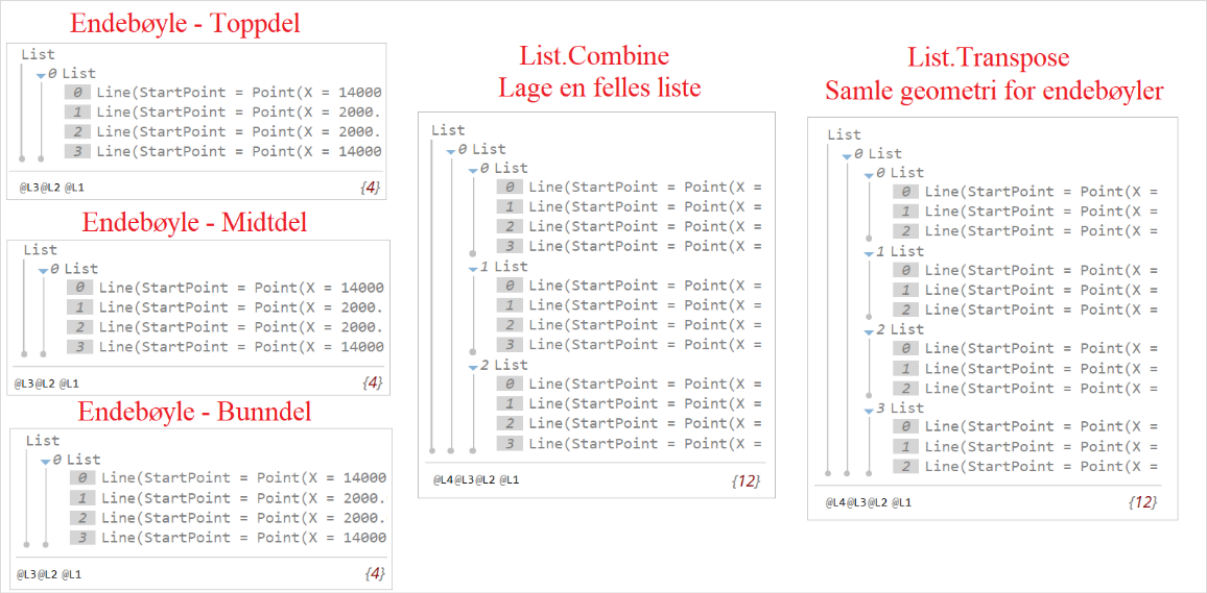
Figur 5.38 Viser hvordan endebøyler lages i Dynamo

Videre etableres en liste som samler hver del, som utgjør en endebøyle. Kombinasjonen ***List.Combine*** og ***List.Create*** benyttes for å samle alle lister i en felles liste. Første punkt fra hver liste blir hentet med ***List.Transpose*** som samler dem i en liste. Denne prosessen repeteres for resterende punkt. Dermed samles tilhørende toppdel, midtdel og bunndel i en liste som utgjør en endebøyle, vist i *figur 5.39*



Figur 5.39: Samle toppdel, midtdel og bunndel i en felles liste

Denne prosessen presenterest også som listeform i *figur 5.40.*

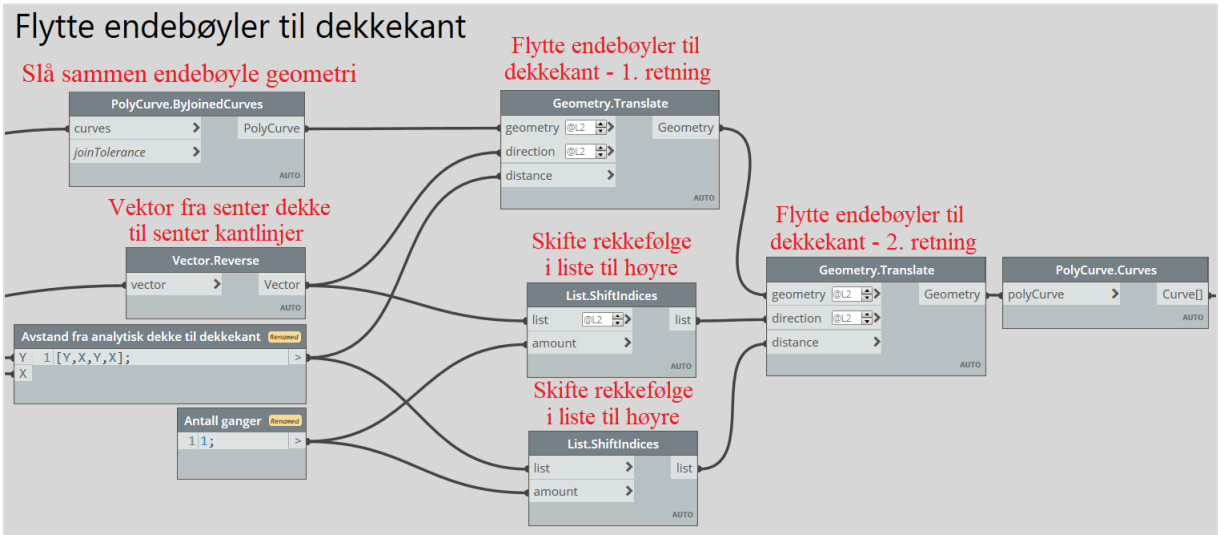


Figur 5.40: Samle toppdel, midtdel og bunndel i en liste illustrert på listeform

5.3.11 Flytte endebøyler fra analytisk linje til dekkekant

Siden analytiske kantlinjer ikke går helt ut til dekkekant, må de flyttes ut. Dette gjøres med å flytte endebøyler først bakover og deretter til siden, slik at de havner i hjørnet av dekke. For å flytte endebøylene bakover brukes ***Vector.Reverse*** for å reveresere vektor fra *figur 5.37.* Altså vil vektoren gå fra sentrum og ut til kantlinjer.

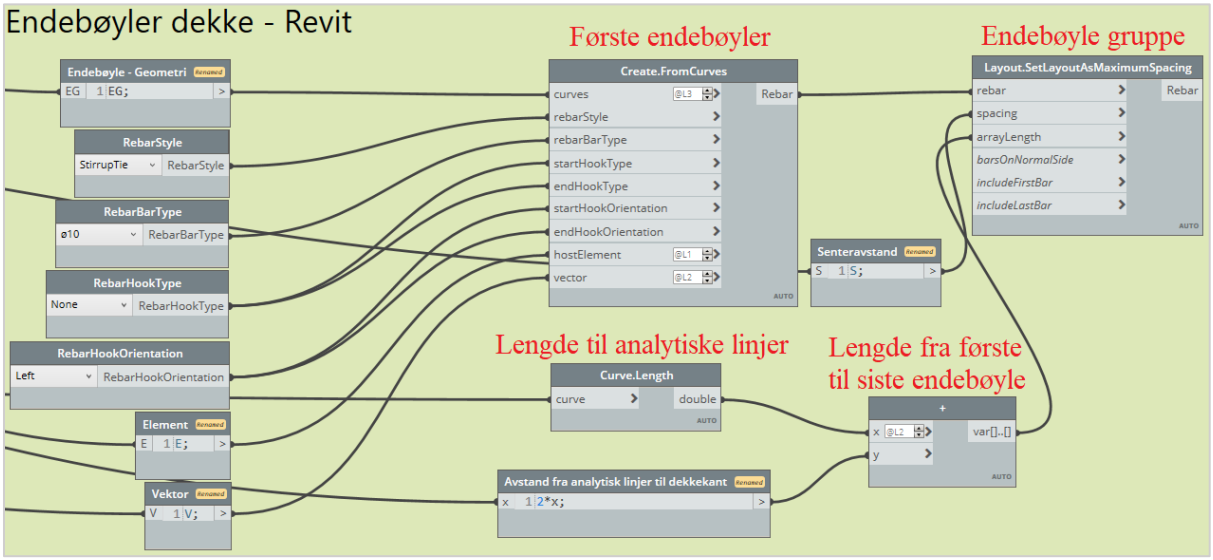
Siden linjer i et rektangel går klokkesvis, benyttes ***List.ShiftIndicies*** for å flytte rekkefølge i en liste frem et punkt. Dette gjøres for listen over avstand og vektor, slik at ***Geometry.Translate*** flytter endebøylene mot hjørnet. Til slutt benyttes ***PolyCurve.Curves*** for å få riktig geometri til overføring, vist i figur *5.41*.



Figur 5.41: Flytte endebøyler fra analytisk linje til dekkekant

5.3.12 Lage gruppe med endebøyler og overføre til Revit

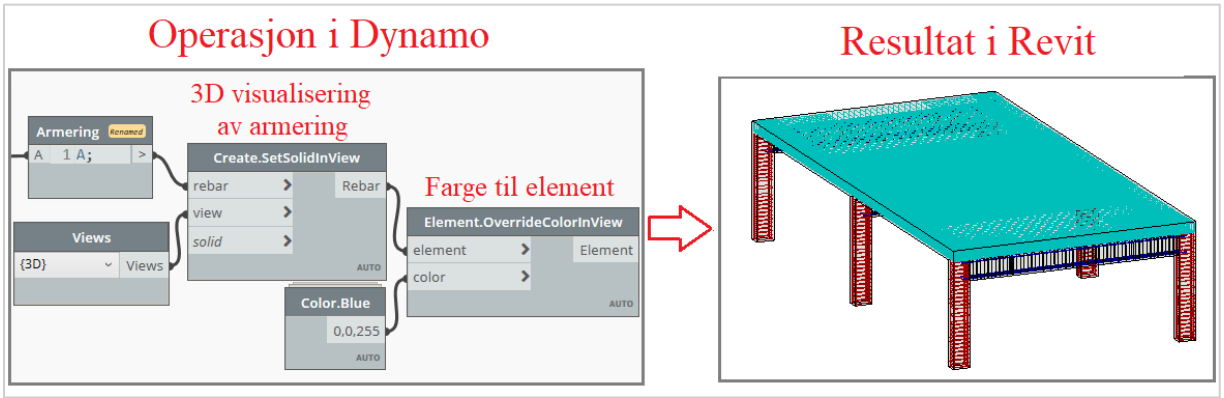
For å lage armering fra geometri brukes ***Create.FromCurves*** som i *delkapittel 5.2.7*. Resterende armeringsjern etableres med ***Layout.SetLayoutAsMaximumSpacing***, der det må oppgis en høyde og avstand fra første til siste endebøyle, vist i *figur 5.42*.



Figur 5.42: Lage endebøyler i Revit

5.4 3D visualisering av armering med farger

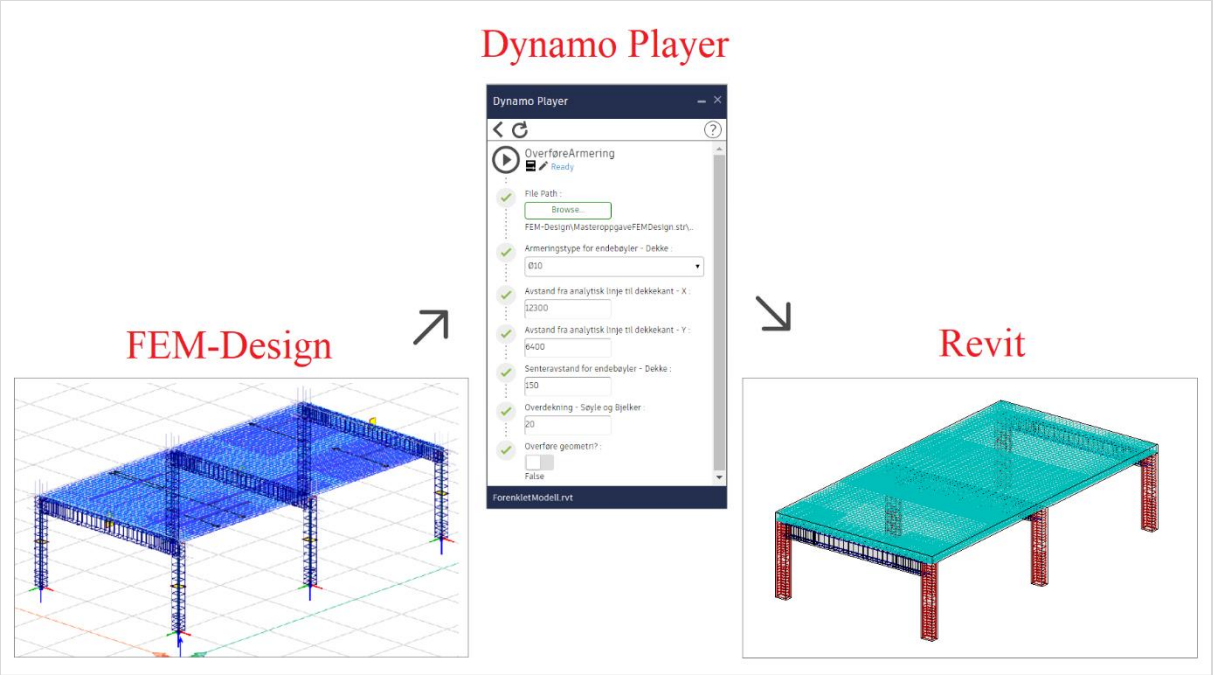
Ofte kan en armeringsmodell virke rotete, noe som fører til at modellen vil være vanskelig å tolke. Det er viktig å skape en visuelt attraktiv presentasjon av armeringsmodellen, slik at det enklere for utførende å tolke modellen. For å bidra til dette formålet brukes ***Create.SetSolidInView***, som visualiserer armering i 3D istedenfor linjer. Videre benyttes ***Element.OverrideColorInView*** for å lage en fargekode som skiller armering i bjelker, søyler og dekke. Operasjonen i Dynamo og resultatet i Revit er illustrert i *figur 5.43.*



Figur 5.43: Visualisere armering i 3D og gi elementer fargekode

5.5 Resultat – Armering i Revit

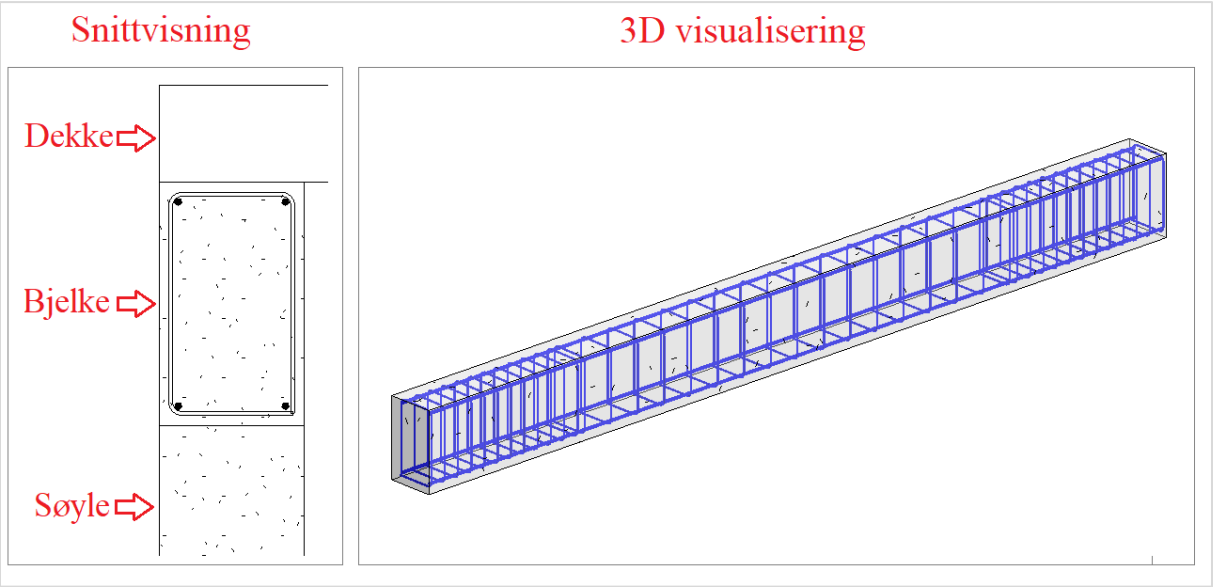
Ved bruk av Dynamo player kan armering i FEM-Design overføres til Revit med bruk av 7 inputs, vist i *figur 5.44*. For å vise hvordan armeringen legger seg i Revit er det delt opp i seksjoner som skiller mellom søyler, bjelker og dekker.



Figur 5.44: Viser hvordan Dynamo Player kan brukes for å overføre en armeringsmodell i FEM-Design til en modell i Revit

5.5.1 Armering i bjelke

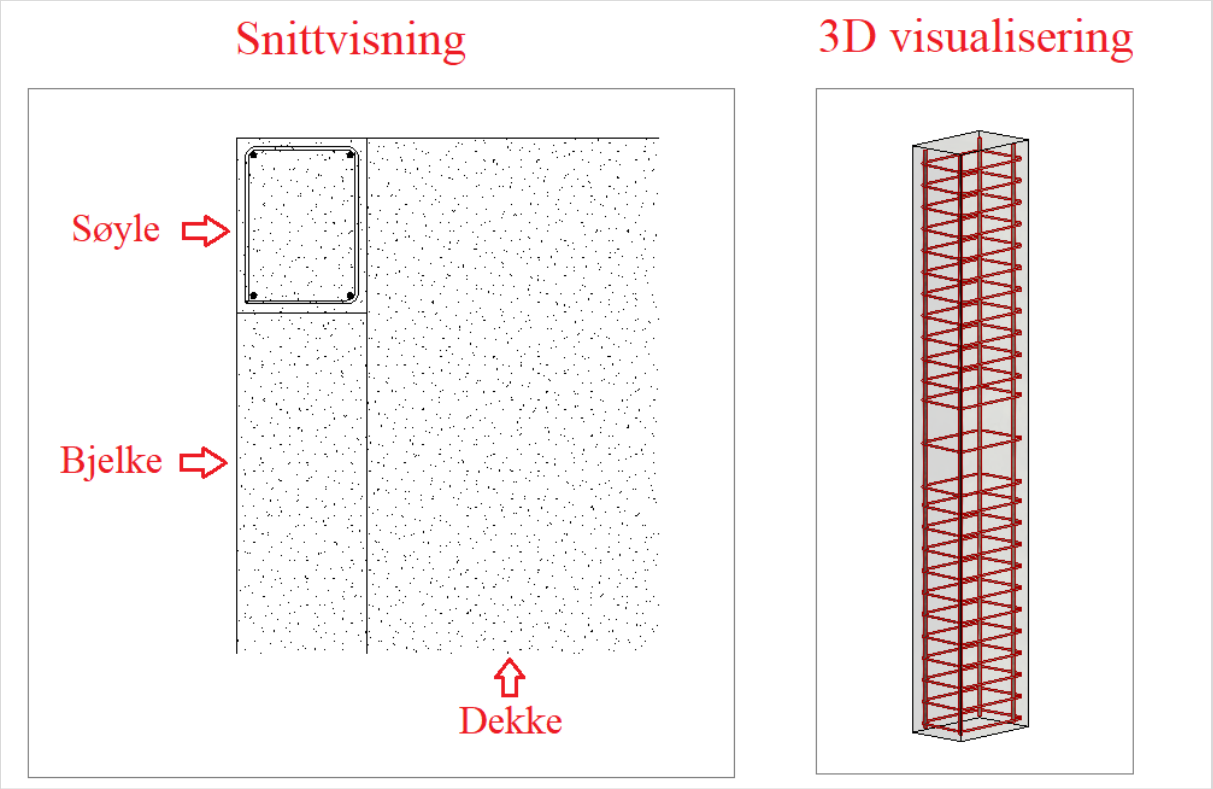
Figur 5.45 viser hvordan armering plasseres i en bjelke i Revit.



Figur 5.45: Viser hvordan armering plasseres i en bjelke i Revit

5.5.2 Armering i søyler

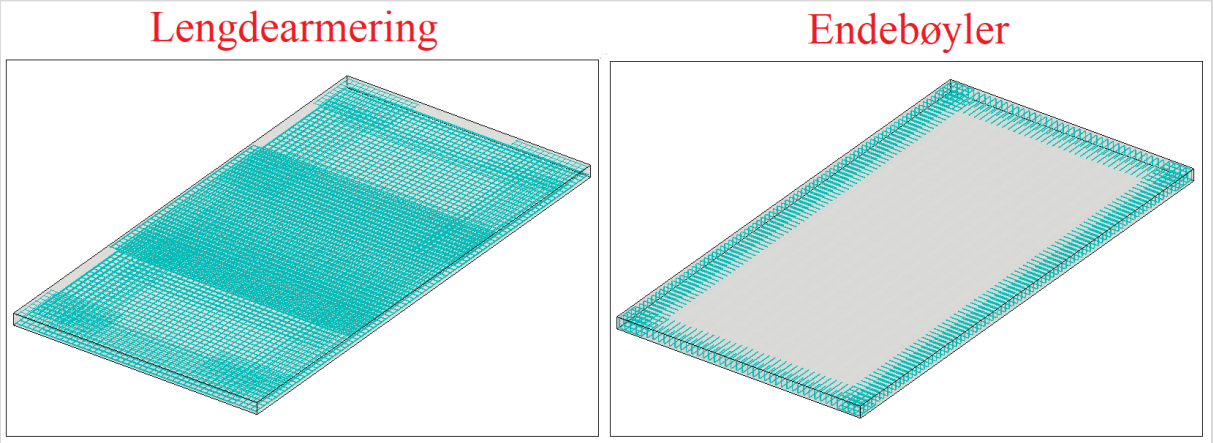
Figur 5.46 viser hvordan armering legger seg i en søyle i Revit.



Figur 5.46: Viser hvordan armering plasseres i en søyle i Revit

5.5.3 Armering i dekke

Figur 5.47 viser hvordan lengdearmering og endebøyler plasseres i Revit.



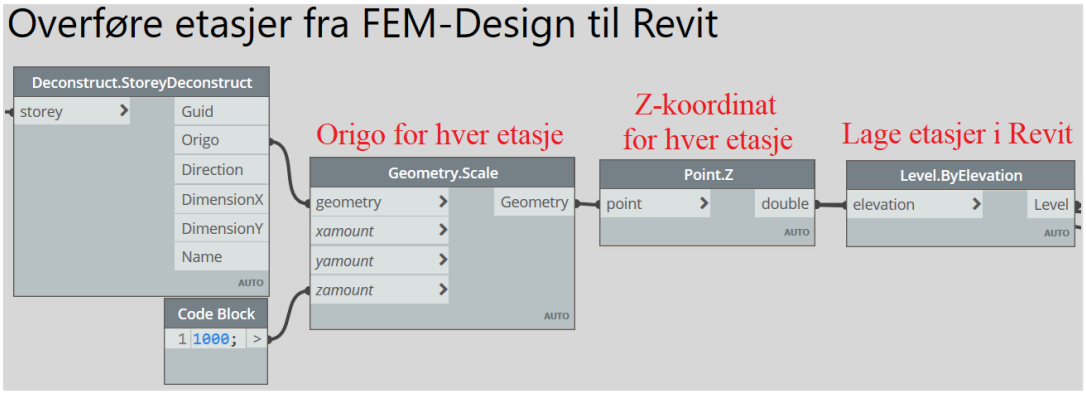
Figur 5.47: Viser hvordan lengdearmering og endebøyler legger seg i Revit

5.6 Overføre elementer fra FEM-Design til Revit

Det er ikke alltid det eksisterer en Revit modell som kan tas utgangspunkt i. I enkelte tilfeller modelleres og beregnes modellen i FEM-Design før den modelleres i Revit. Dermed er det også gunstig å tilrettelegge for en overføring av strukturelle elementer fra FEM-Design til Revit. Overføringen inkluderes i scriptet hvor armering overføres, da det er mange felles noder som benyttes.

5.6.1 Overføre etasjer

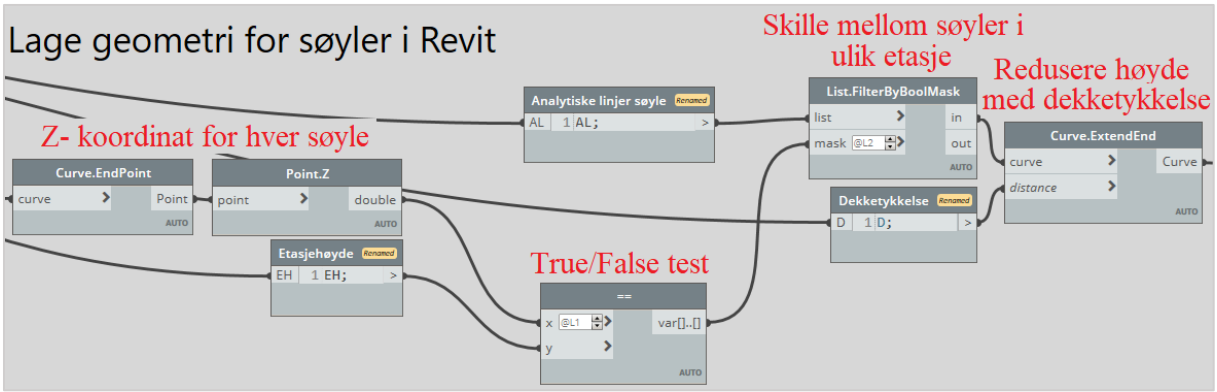
For organisering av en modell er det alltid en fordel å etablere etasjer i en Revit modell og dette gjøres ved å ta utgangspunkt i eksisterende etasjer i FEM-modellen. ***Deconstruct.StoreyDeconstruct*** brukes for å hente origo for hver etasje. Høyden til hver etasje finnes med ***Point.Z*** og ***Level.ByElevation*** etablerer til slutt etasjene i Revit basert på høydene som oppgis, vist i *figur 5.48*.



Figur 5.48: Overføring av etasjer fra FEM-Design til Revit

5.6.2 Overføre søyler

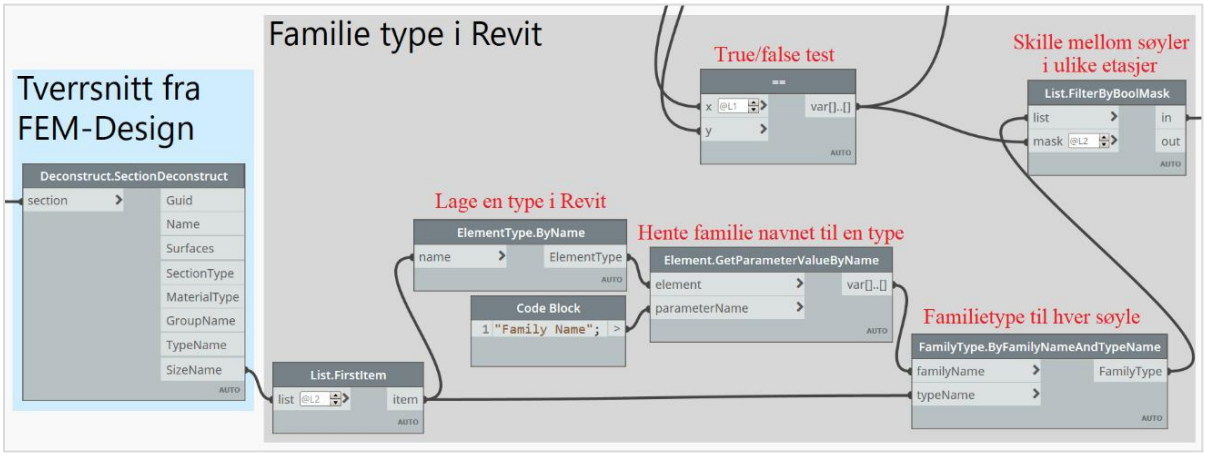
Som utgangspunkt for å lage søyler i Revit brukes søyler sine analytiske linjer fra FEM-modell. Kombinasjonen ***Curve.EndPoint*** og ***Point.Z*** gir høyeste Z-koordinaten til hver søyle. Denne brukes i ***==*** sammen med etasjehøyder som skaper en true/false test. Testen brukes i ***List.FilterByBoolMask*** som lager en liste som skiller søyler i ulike etasje. For å ta hensyn til at søyler ikke modelleres utifra analytiske linjer i Revit, reduserest høyde med overliggende dekketykkelse, vist i *figur 5.49*. Ved å skille elementer i etasje, oppnås det også at en søyle sin høyde alltid reduserest med overliggende dekke.



Figur 5.49: Tilpasse geometrien til søyle

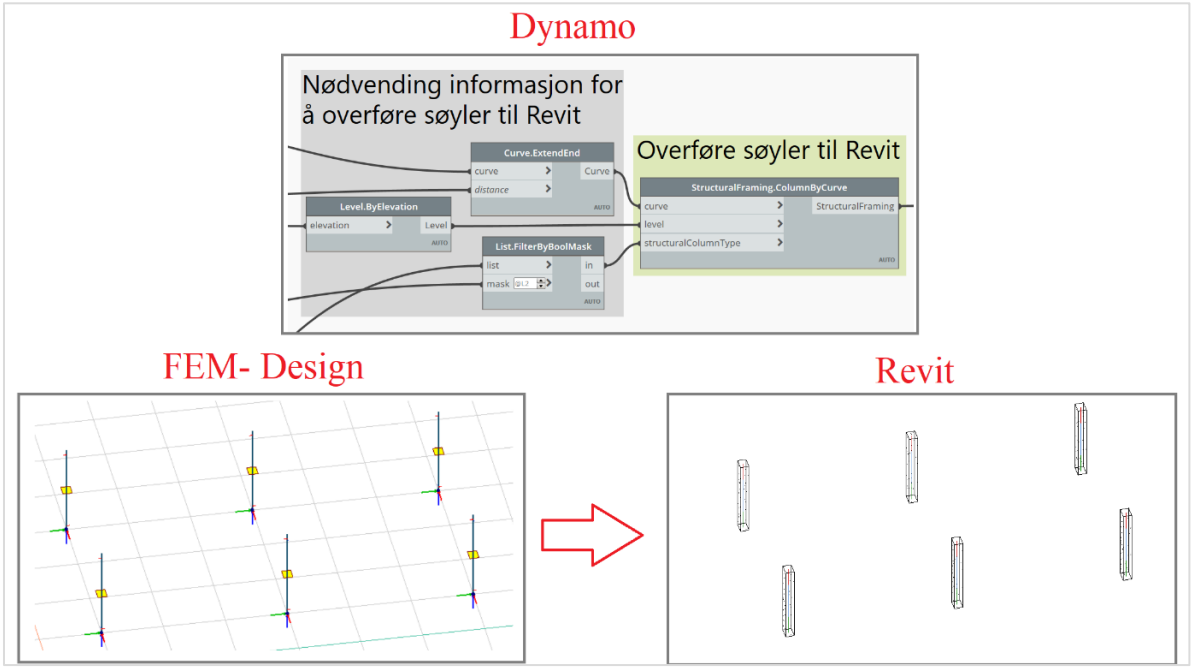
Videre må hver enkelt søyle gis en *familytype* i Revit. ***Deconstruct.SectionDeconstruct*** sin output, *SizeName*, gir dimensjonene til en søyles tverrsnitt i FEM-Design, og er utgangspunktet for å definere en familytype. Dimensjonene kobles på ***ElementType.ByName*** som lager en type basert på et navn. Det spesifiseres at navnet til en *type* i Revit må korrespondere med tverrsnittnavnet i FEM-Design.

For å finne familien til en type brukes ***Element.GetParameterValueByName*** som henter en parameter til et element**.** ***FamilyType.ByFamilyNameAndTypeName*** benyttes for å kombinere en type og tilhørende familie til en familytype. Til slutt må det etableres en tilsvarende listestruktur som for søyle geometri ved bruk av ***==*** og ***List.FilterByBoolMask.*** Prosessen er vist i figur 5.50



Figur 5.50: Lage en familietype i Revit basert på et tverrsnitt i FEM-Design og skille disse i tilhørende etasjer

Da er relevant informasjon definert for å lage en søyle i Revit med ***StructuralFraming.ColumnByCurve***. Hvordan denne prosessen er i Dynamo, FEM-Design og Revit er illustrert i *figur 5.51.* Som tidligere er bjelker utført på tilsvarende metode og vil ikke bli gjennomgått.

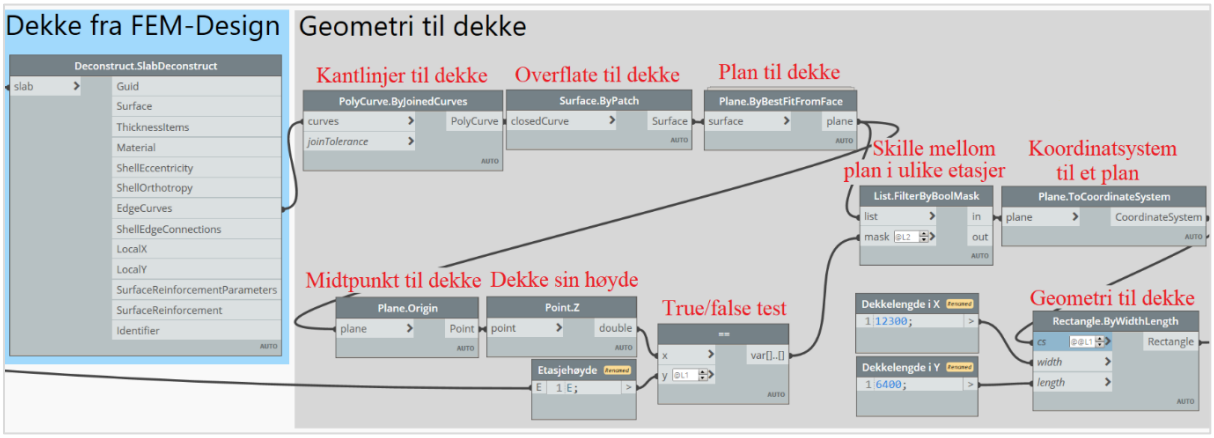


Figur 5.51: Lage søyler i Revit og en illustrasjon som viser søyler i FEM-Design til søyler i Revit

5.6.3 Overføre dekke

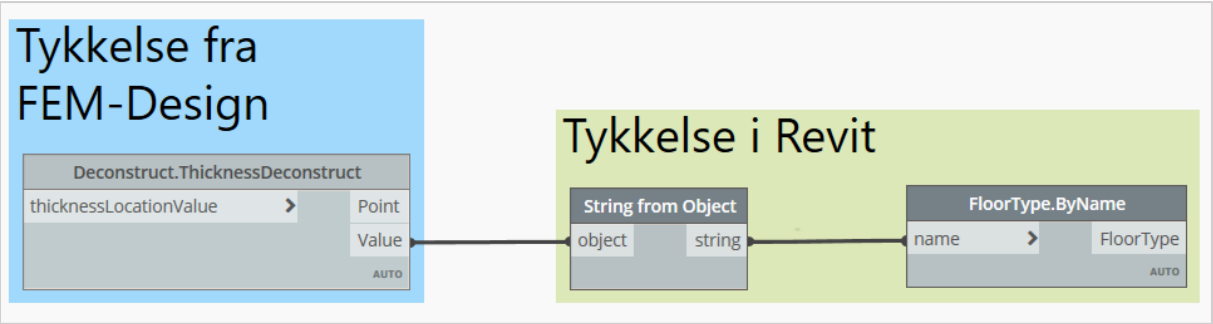
For å overføre et dekke til Revit tar det utgangspunkt i kantlinjer til et dekke fra FEM-Design modell. Kantlinjene hentes med ***Deconstruct.SlabDeconstruct*** som har *EdgeCurves* som output. For å samle goemetrien brukes ***PolyCurve.ByJoinedCurves*** og ***Surface.ByPatch*** lager en overflate utifra samlet geomtri. Basert på en overflate defineres et plan med ***Plane.ByBestFitFromFace***. Videre brukes ***Plane.Origin*** for å finne midtpunktet til dekke og ***Point.Z*** gir dekke sin høyde.

Deretter benyttes **==** for å etablere en true/false test mellom etasjehøyder og dekke sin høyde for å finne et dekke sin tilhørende etasje. Denne testen brukes som betingelse for å skille mellom dekkeplan i ulike etasjer med ***List.FilterByBoolMask*** som videre brukes til å definere et koordinatsystem med ***Plane.ToCoordinateSystem***. Til slutt lages geometrien til dekke med ***RectangleByWidthLength***, der det er behov for å oppgi bredde og lengde til dekke, vist i *figur 5.52*. Grunnen til at dette må oppgis er at FEM-Design bruker analytiske kantlinjer som ikke går helt ut til dekkekant i Revit, som nevnt i *delkapitel 5.3.5.*



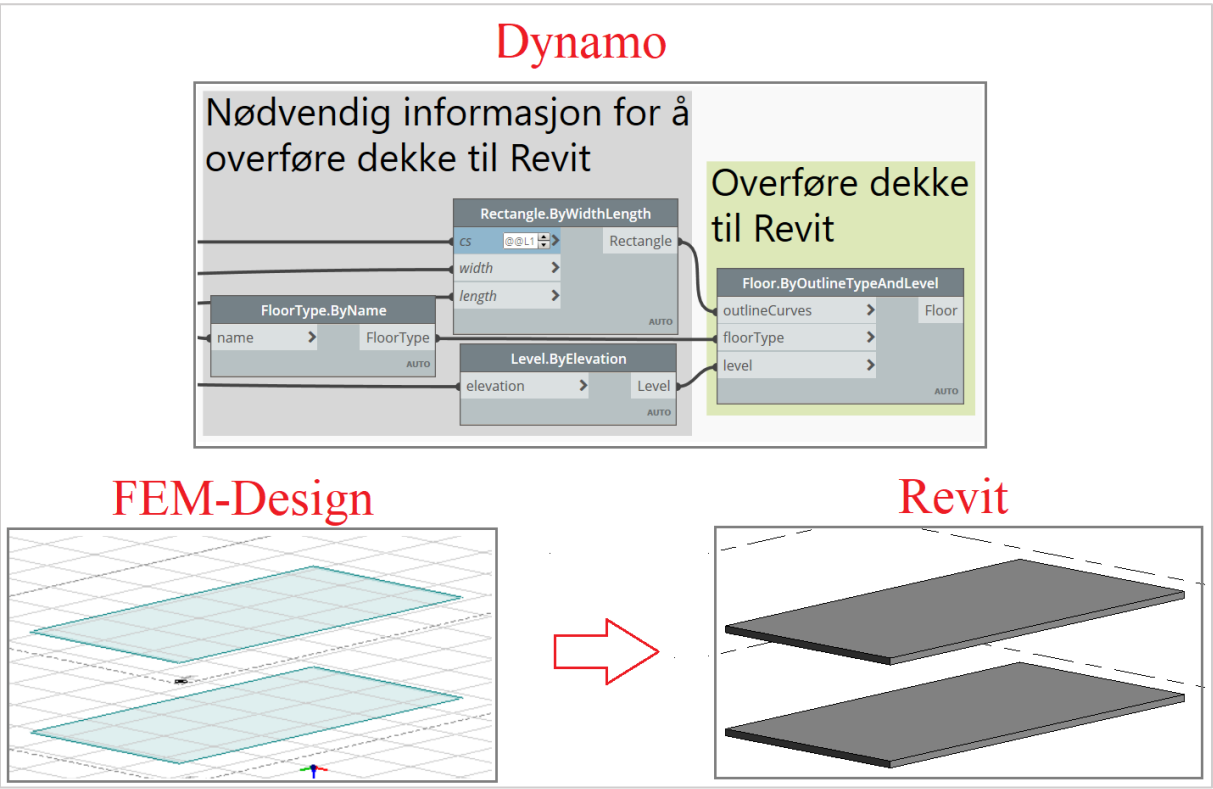
Figur 5.52: Lage geometri til dekke og skille dekker i tilhørende etasjer

Videre defineres en *FloorType* for dekke, vist i *figur 5.53*. Dette gjøres ved å hente tykkelsen til dekke fra FEM-Design med ***Deconstruct.ThicknessDeconstruct***. ***String from Object*** brukes for å gjøre et objekt om til tekst og ***FloorType.ByName*** velger en *FloorType* fra Revit basert på et navn. For at operasjonen skal fungere er det nødvendig at navnet på en *FloorType* i Revit er gitt som dekketykkelse, som for eksempel «200».



Figur 5.53: Lage en FloorType i Revit

Relevant informasjon er funnet, og med bruk av ***Floor.ByOutlineTypeAndLevel*** lages dekke i Revit**.** Hvordan denne prosessen er i Dynamo, FEM-Design og Revit er illustrert i *figur 5.54*.



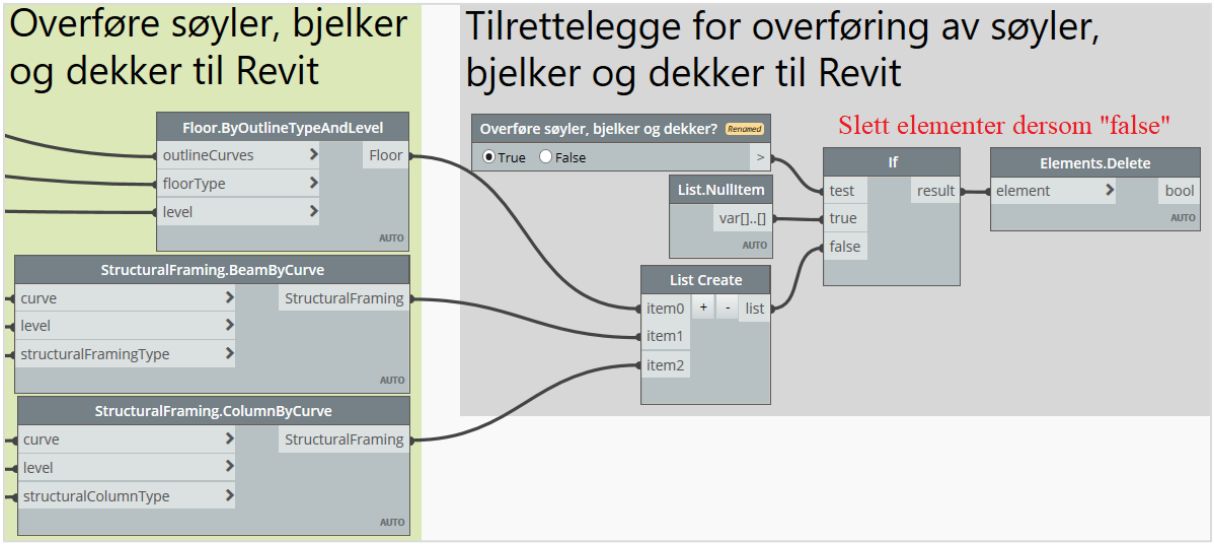
Figur 5.54: Lager et gulv i Revit og viser en illustrasjon i hvordan dekke ser ut i FEM-Design og i Revit

5.6.4 Tilrettelegge for et valg om å overføre strukturelle elementer

Det er ikke alltid det er ønskelig å overføre strukturelle elementer sammen med armering fra FEM-Design. Derfor er det nødvendig å tilrettelegge for at brukeren har et valg som gjør det mulig å kun overføre armering.

For å oppnå dette brukes ***If***, som gir et resultat basert på om en test er *true* eller *false.* Her må det oppgis en ***Boolean*** som er en klikkbar true/false test, samt et *true* og *false* utsagn. Som et *false*- utsagn brukes en liste der alle strukturelle elementer er samlet med ***List Create***. ***List.NullItem*** brukes som et *true* utsagn, som vil gi en liste med nullverdier.

Resultatet av testen kobles videre på en ***Element.Delete*** som vil slette alle elementer som er koblet til denne noden. Dersom *true* velges vil ***Element.Delete*** slette en liste med nullverdier, altså vil strukturelle elemente laged med armering Velges *false* vil derimot ***Element.Delete*** slette alle strukturelle elementer som er hentet fra FEM-Design, og det er kun armering som blir laget. Prosessen er vist i *figur 5.55*.

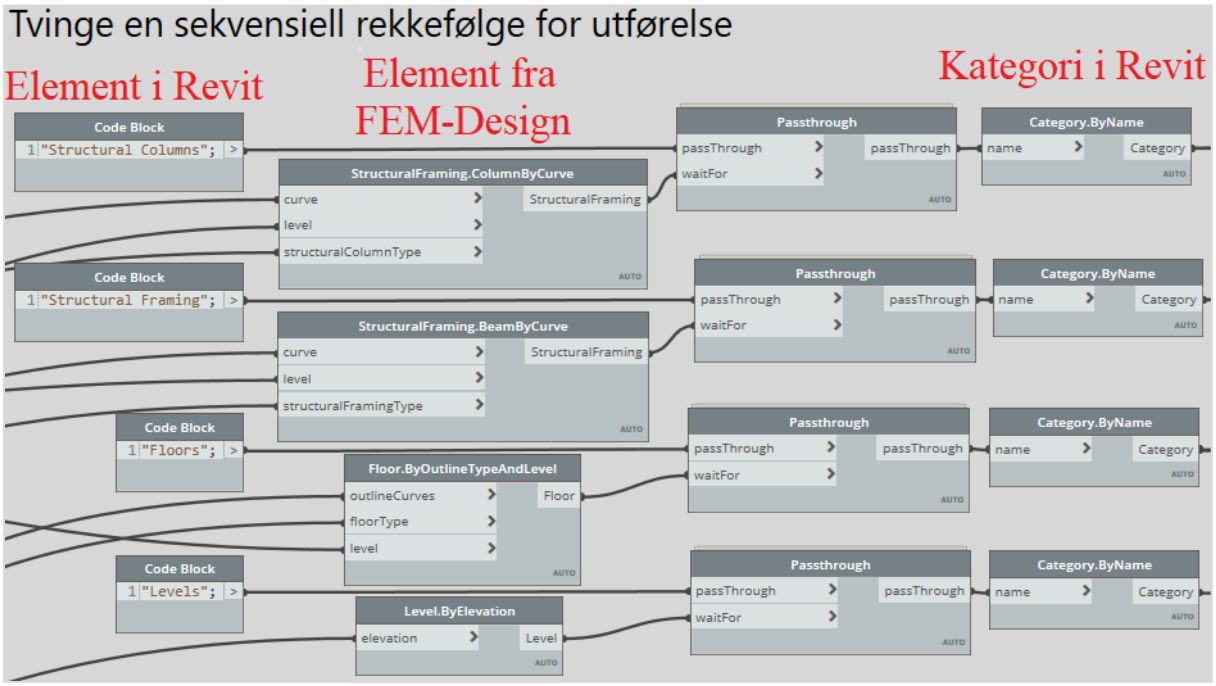


Figur 5.55: Tilrettelegge et valg om å overføre søyler bjelker og dekker til Revit

5.6.5 Tvinge en sekvensiell utførelse av prosesser

Når et script kjøres, vil alle prosesser i scriptet kjøre samtidig. Dersom en prosess er avhengig av at en annen er utført vil det oppstå en feilmelding. Dette er en problemstilling som oppstår ved overføring av armering sammen med strukturelle elementer. Armering er laget basert på at det allerede eksisterer strukturelle elementer i Revit. Når disse overføres sammen med armering, eksisterer ingen strukturelle elementer når armeringen skal lages, noe som vil resultere i en feilmelding.

Dette løses med en ***Passthrough*** node, som tvinger en sekvensiell utførelse av prosesser. Noden som er koblet på *waitFor* inngangen blir utført først og deretter utføres noden som blir koblet på *passThrough* inngangen. Altså vil strukturelle elementer overføres til Revit, før Dynamo henter informasjon om disse og plasserer ut armering, vist i *figur 5.56*.



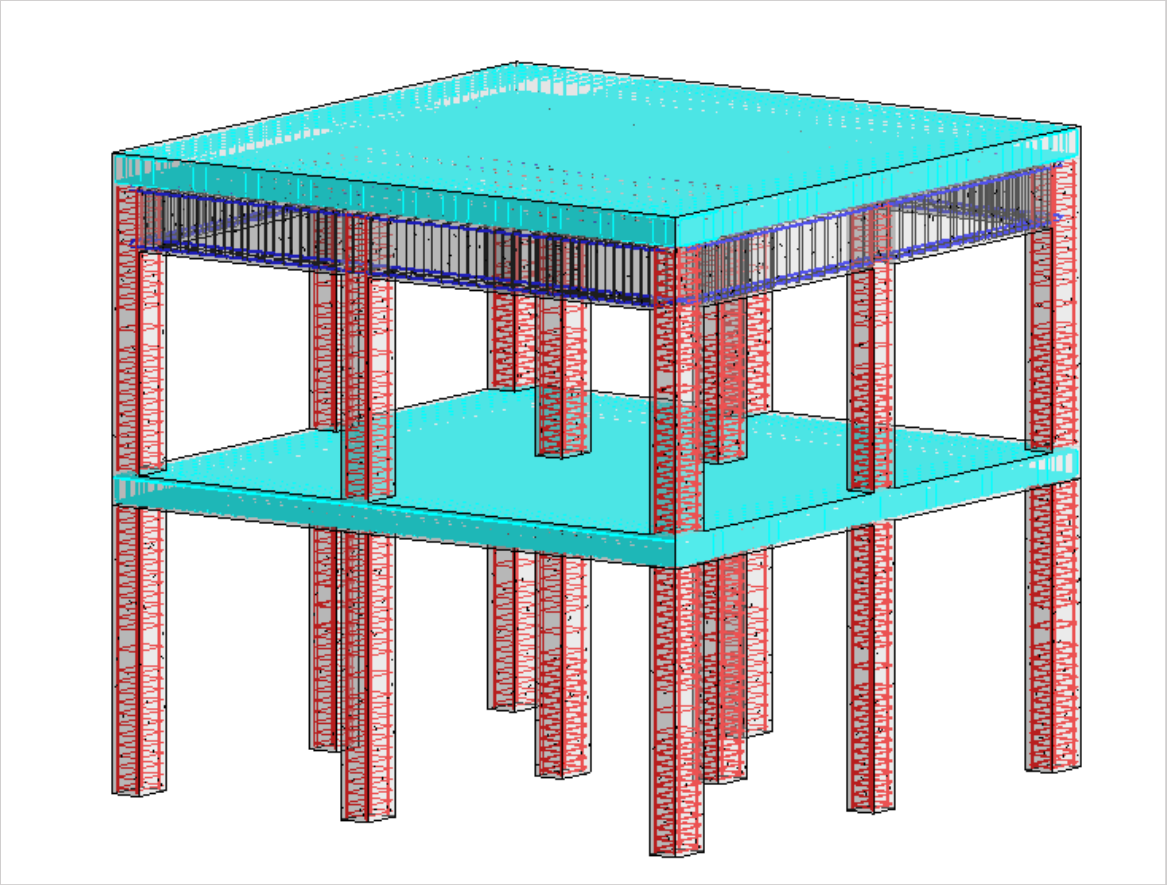
Figur 5.56: Tvinge en sekvensiell rekkefølge for utførelse

5.7 Validering av script

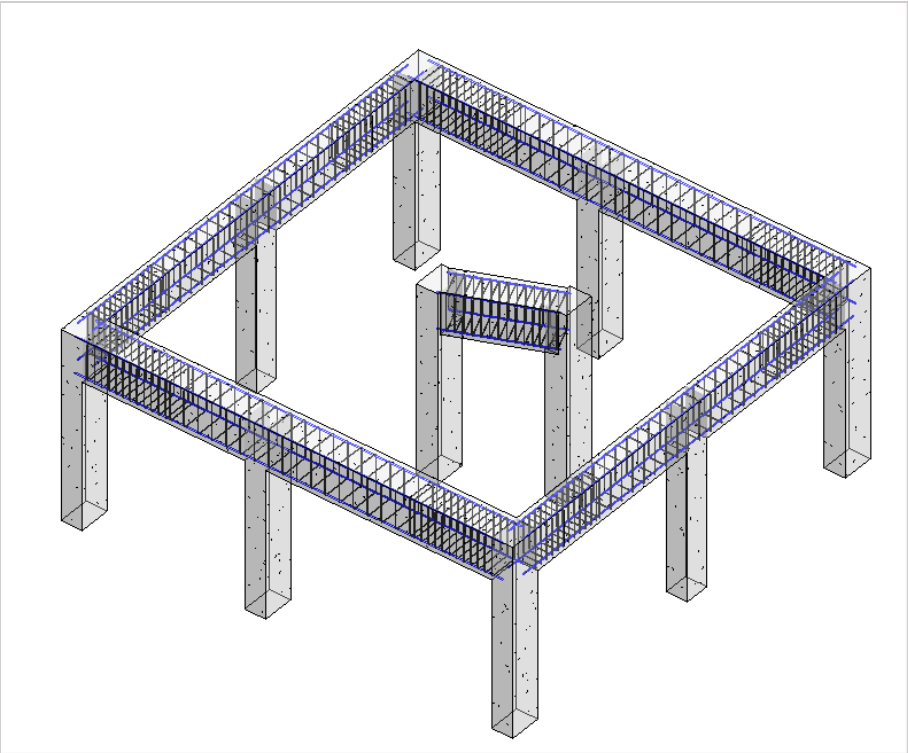
For at et script skal være av verdi er det gunstig at scriptet er fleksibelt slik det kan anvendes ved flere prosjekter. Siden det er en tidkrevende prosess å utarbeide et script, er det sannsynligvis ikke tidsbesparende dersom det kun skal brukes til et prosjekt. Derfor er det viktig å lage en plan over hva som er ønsket av scriptet og hvordan man kan gjøre det så fleksibelt som mulig.

For å oppnå et fleksibelt script er det nødvendig å hele tiden lete etter sammenhenger mellom elementer. Her er formler i *Code Blocks* og innhenting av parameterverdier sentrale metoder som brukes til å uttrykke sammenhenger.

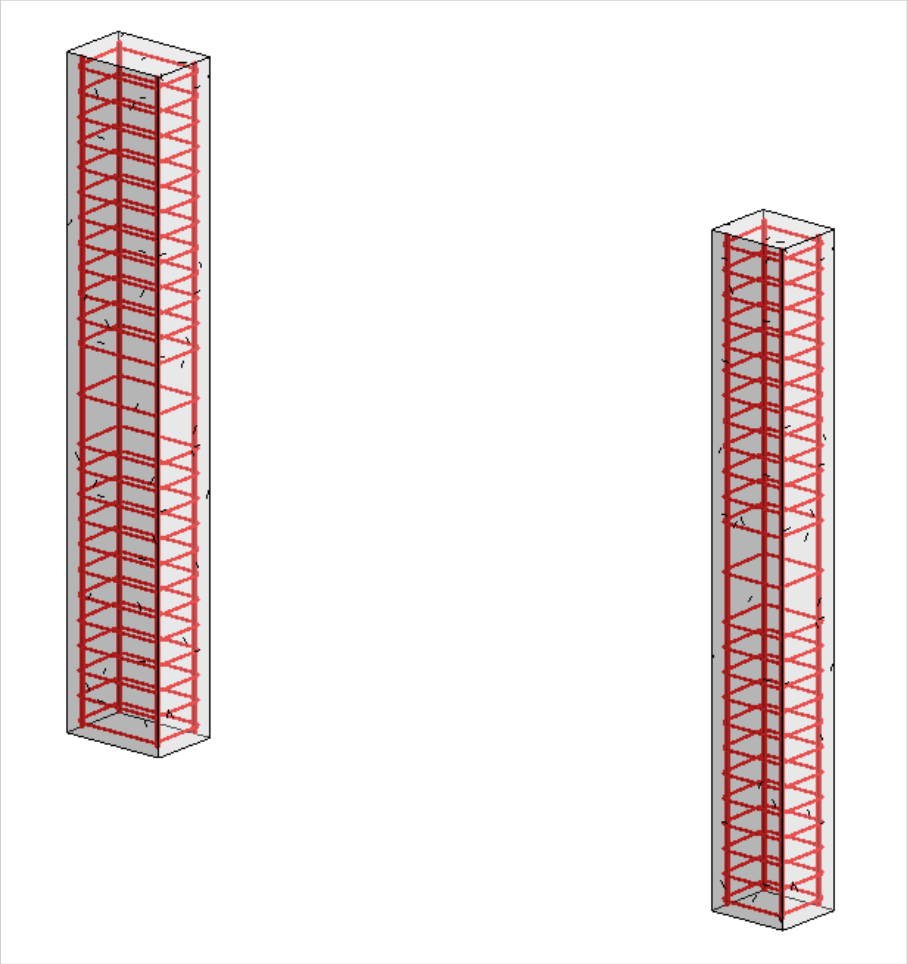
For å kunne validere at scriptene kan anvendes for flere prosjekter, er scriptene testet ut gjennom hele utviklingsprosessen. Ved endringer og nye prosesser i scriptet, kjøres scriptet på nytt for å være sikker på at resultatet er tilfredstillende. Scriptet lages med utgangspunkt i en forenklet modell som forklart i *delkapittel 3.3*. Videre er scriptet testet på andre modeller for å sikre at det fungerer blant annet for flere etasjer, skråbjelker og ulike tverrsnitt. Eksempel på dette er vist i *figur 5.57, 5.58 og 5.59*.



Figur 5.57: Viser at scriptet fungerer for flere etasjer



Figur 5.58: Viser at scriptet fungerer for alle retning i XY plan, samt skråbjelker



Figur 5.59: Viser at scriptet fungerer for ulike tverrsnitt

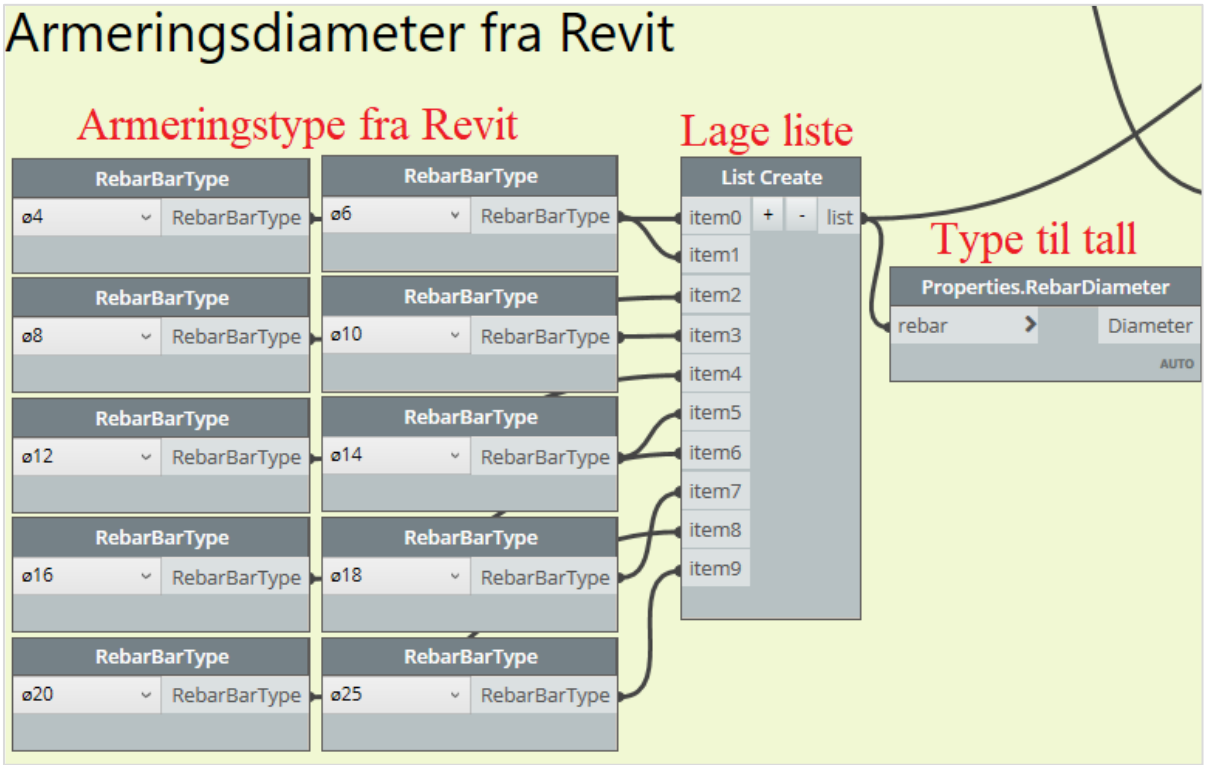
6 Diskusjon

Hovedmålet for oppgaven har vært å utforske hvordan Dynamo kan brukes til å automatisk overføre elementer mellom Revit og FEM-Design. Gjennom oppgaven har store deler av fokuset vært på en overføring av armeringsmodell fra FEM-Design til Revit. Dette kapitellet vil diskutere kvalitet til scriptene, samt vurdere forutsetninger og begrensninger. Siden det eksisterer en tilsvarende overføring i StruXML, vil også scriptene sammenlignes med dette verktøyet.

6.1 Forutsetninger for at script skal kjøres

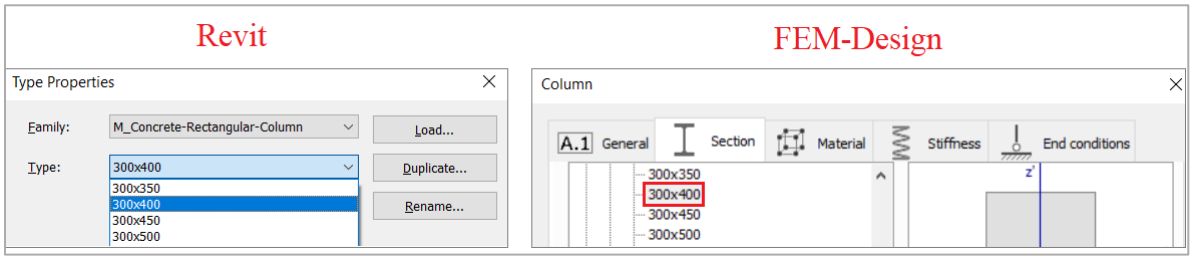
Når det utarbeides et script, er det ønskelig å automatisere flest mulig operasjoner. For å muliggjøre dette, kan det oppstå situasjoner der scriptet er avhengig av noen forutsetninger for å kjøres. Scriptene i denne oppgaven inneholder noen forutsetninger som blir gjennomgått i dette delkapitellet.

For å etablere en sammenheng mellom en armeringsmodell i FEM-Design og Revit er det nødvendig at eksakt armeringsdiameter korresponderer i begge program. Når scriptene brukes for første gang, må brukeren inn i selve scriptet og laste opp sin egen armeringsfamilie, vist i *figur 6.1*. Dette er fordi armeringsfamilien i en Revit modellen kan variere hos person til person.



Figur 6.1: Armeringstype fra Revit

Det er også nødvendig at tverrsnittnavn i FEM-Design stemmer overens med *type* navn i Revit, som vist i *figur 6.2*. Denne forutsetningen er noe tidkrevende da det må etableres et nytt *type* bibliotek i Revit, for å få navn til å samsvare. Dette vil stille et større krav til antall ganger scriptet skal brukes for at det skal kunne anses som tidsbesparende.



Figur 6.2: Viser at et tverssnitt i FEM-Design må koorespondere med en type i Revit

6.1.1 Pakker i Dynamo

Det er laget en oversikt over noder som er brukt i scriptet, hvilken pakke den tilhører og en kort forklaring. Denne er lagt med som vedlegg i *delkapittel 10.1*. Siden Dynamo sin egen database ikke inneholdt noder som kunne utføre spesifikke operasjoner, var det nødvendig å laste ned aktuelle nodepakker fra biblioteket. Dersom pakker som benyttes i scriptet ikke er lastet ned, vil det oppstå feimelding når scriptet kjøres. Følgende pakker er brukt i scriptet:

- archi-lab.net
- Clockwork for Dynamo 2.x
- FemDesign
- Structural Design

6.2 Begrensninger ved parametrisk design

Til tross for alle fordeler knyttet til parametrisk design, vil det alltid eksistere noen begrensninger. Et script inneholder ofte mange grupper og noder, slik at det vil være komplisert for en tredjepart å finne frem i scriptet og gjøre endringer. Enhver utvikler av et script har sine egne tanker og ideer om hvordan scriptet skal settes sammen for å oppnå et best mulig resultat. Dette fører til at et hvert script består av en unik organisering og struktur.

Sammenhenger mellom noder vil ofte føre til å styrke intelligensen i et script, men kan også være begrensende i enkelte situasjoner. En endring av en funksjon tidlig i scriptet kan ha en enorm effekt på de påfølgende funksjonene, noe som kan føre til at scriptet ikke fungerer som det skal. Dette gjelder spesielt for en tredjepart som skal gjøre en liten endring, men ikke forstår at endringen vil skape en domino effekt. Denne risikoen reduseres med en god oversikt og organisering av scriptet, men vil alltid være en begrensning til parametrisk design.

6.2.1 Begrensninger ved foreslått script

Å lage et script som overfører en modell fra Revit til FEM-Design, samt utfører beregninger, lager en armeringsmodell i FEM-Design og til slutt overføre armering tilbake til Revit er essensen i oppgaven. At scriptet skal fungere for enhver modell er veldig omfattende arbeid og for tidkrevende for denne oppgaven. Det er derfor innført noen avgrensninger i oppgaven som presenterest.

Første avgrensning for oppgaven er at scriptet er laget for rektangulære bjelker, søyler og dekke. Dette er gjort for å skape en forenklet modell som representerer en realistisk arbeidssituasjon, samt unngår mye repetativt arbeid. Det er selvsagt få modeller som består av kun denne typen geometri. Å tilrettelegge for all tverrsnitt geometri kunne vært prioritert, og ville gjort scriptet mer attraktivt, men likevel viser oppgaven potensialet til parametrisk design.

Scriptet i seg selv er ikke stort nok til at det er tungt å kjøre, men kan bli et problem dersom det benyttes på store modeller. Dette er fordi en større modell inneholder flere elementer, som resulterer i flere operasjoner som utføres samtidig. En annen begrensning er at scriptet ikke lager armering mellom overganger fra søyler til bjelker. Dermed er dette noe som brukeren må gjøre selv. Grunnen til at det ikke er inkludert er at overgangen ikke er gjort i FEM-Design. Det er også veldig varierende, slik at det er vanskelig å automatisere dette.

6.3 StruXML Revit add-in

Strusoft, produsent av FEM-Design, har sin egen add-in i Revit som tilbyr en toveis kommunikasjon mellom programmene. For å kunne oppnå en overføring mellom programmene er det et krav om at Revit modellen inneholder analytiske elementer. Da er det mulig å overføre analytiske elementer og tilhørende egenskaper mellom programmene.

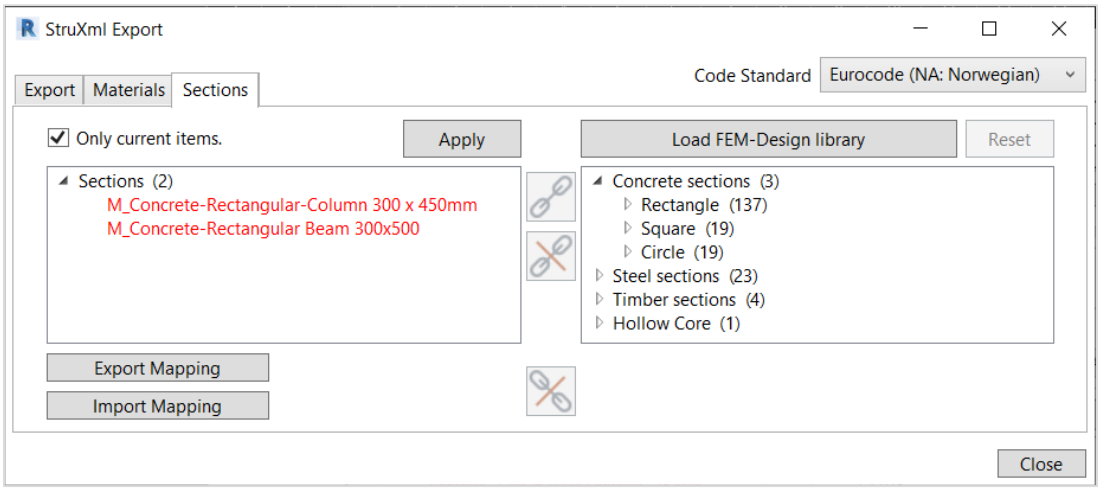
StruXML utfører en tilsvarende operasjon som scriptene i oppgaven, og dermed er det naturlig å sammenligne disse. Dette delkapitellet tar for seg hvordan StruXML brukes, samt begrensninger og sammenligning med script.

6.3.1 Bruk av StruXML add-in

Før en overføring er det viktig å korrigere og validere den analytiske modellen i Revit. Den eksakte analytiske modellen overføres til FEM-Design, og er derfor nødvendig at den er korrekt. Ved bruk av StruXML sin overføring er det behov for å gjøre noe forhåndsarbeid, som presenteres i dette delkapittelet.

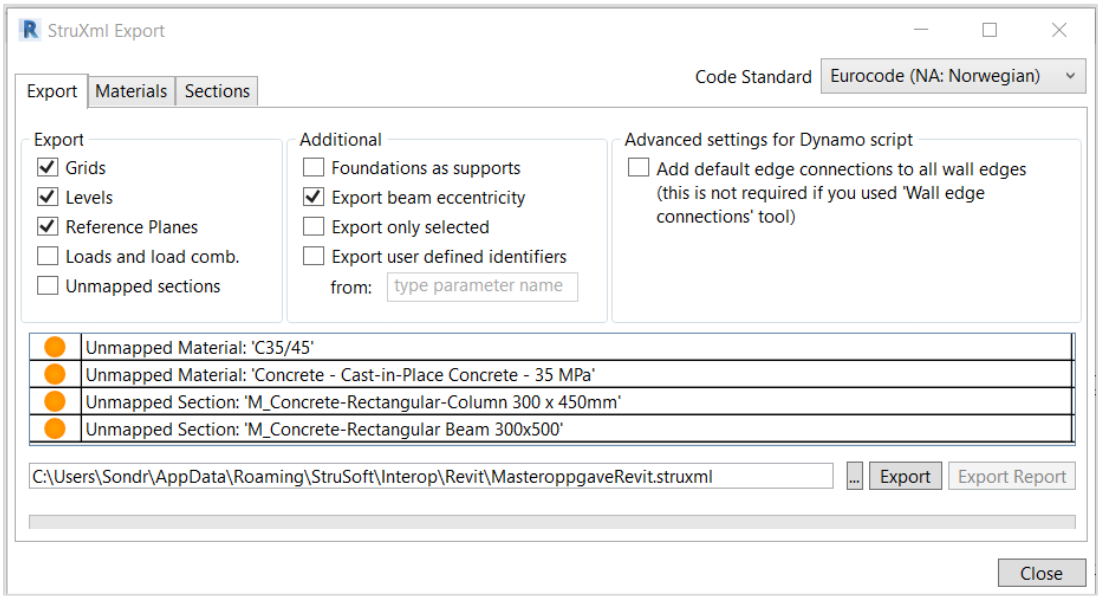
6.3.1.1 Eksport av Revit modell til FEM-Design

StruXML sitt eksport verktøy består av tre faner: Eksport, materialer og tverrsnitt. Når verktøyet åpnes oppstår det en feilmelding som sier at materialer og tverrsnitt er «*unmapped*», vist i *figur 6.3.* Dette betyr at tverrsnitt i Revit modellen ikke korresponderer til tverrsnitt i FEM-Design. For å få dem til å korrespondere må det velges et tverrsnitt i FEM-Design som tilsvarer tverrsnitt i Revit. Dette må gjøres for hvert tverrsnitt, og samme prosess gjentas også for materialer.



Figur 6.3 Viser hvordan tverrsnitt fanen ser ut i StruXML

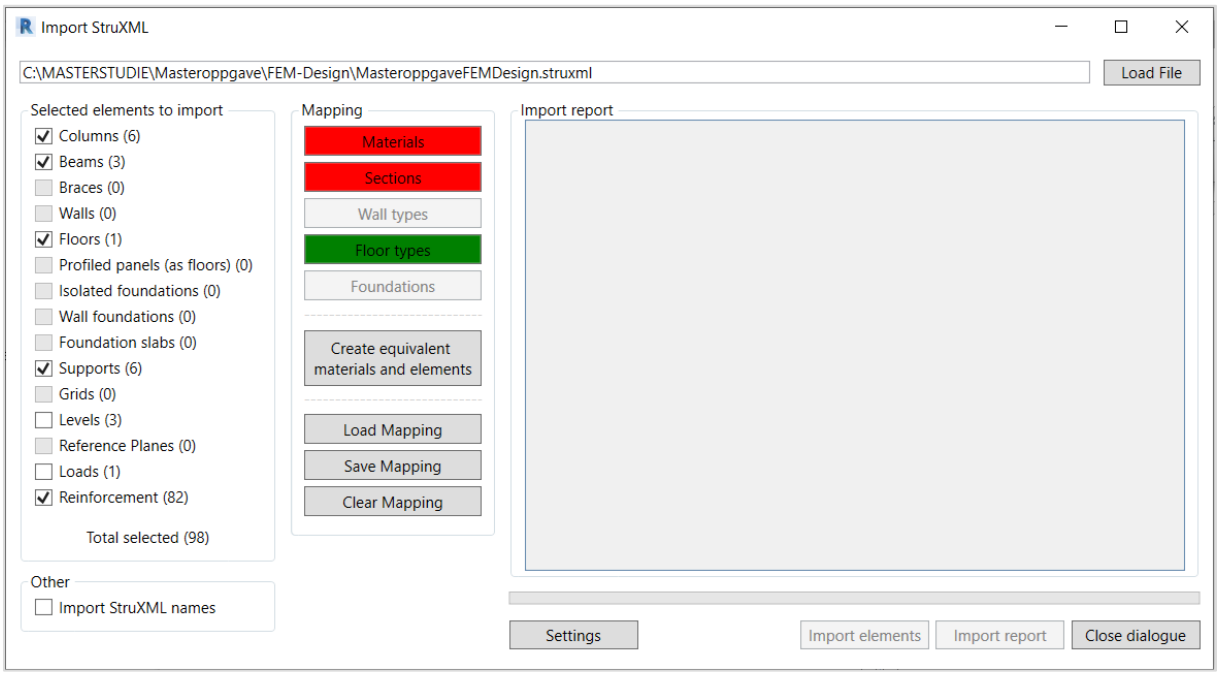
Eksporten kan utføres når det ikke eksisterer flere feilmeldinger, som gjøres i eksport fanen. Brukeren har mulighet å velge mellom hva som skal overføres, vist i *figur 6.4*. Her kan det hukes av for eksport av bjelkers eksentrisitet, som er viktig dersom modellen skal overføres tilbake til Revit igjen.



Figur 6.4 Viser hvordan eksport fanen ser ut i StruXML

6.3.1.2 Import av FEM-Design modell til Revit

Tilsvarende som for eksport er det nødvendig at material/tverrsnitt fra FEM-Design korresponderer med material/tverrsnitt i Revit. Når det er fikset, velges elementer som er ønsket å ta med i importen. *Figur 6.5* viser hvordan import verktøyet til StruXML ser ut.



Figur 6.5 Viser hvordan import verktøyet til StruXML ser ut

6.3.1.3 Begrensninger med StruXML

Et problem med verktøy som praktiserer automatisk overføring er at det ofte oppstår problemer relatert til analytiske linjer. Dette er noe som også er nevnt i brukermanualen til StruXML: «Armering importeres til Revit uten noen modifikasjoner. Det betyr at den eksakte armeringen fra FEM-Design blir gjenskapt i Revit, og dermed er det ofte behov for å korrigere armeringen i Revit» (Bjergø, Reinforcement, 2020)

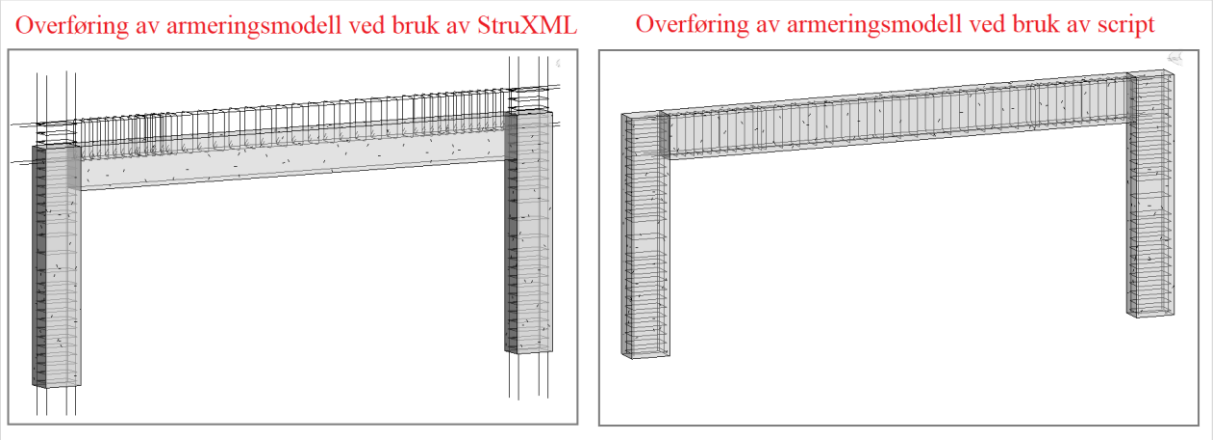
Etter å ha kontaktet forfatter av brukermanuelen angående dette problemet, ble det bekreftet at dette ofte er et problem, og kan løses med eksentrisitets innstillinger (Bjergø, 2022). Dersom brukeren er konsekvent med å korrigere eksentrisitet innstillinger i Revit og FEM-Design skal armering overføres med riktig plassering i Revit. Å holde styr på eksentrisitet til hvert element kan være tidkrevende og dermed er det gunstig å lage et script som tar hensyn til dette.

6.3.2 Sammenligning mellom foreslått script og StruXML

StruXML utfører en tilsvarende overføring som scriptene. Dette skaper en mulighet for å bruke StruXML til å sammenligne med resultatene som oppnås med scriptene. Som nevnt i *delkapittel 6.3.1.3* oppstår det ofte problemer med eksentrisitet til et analytisk element. Ved utarbeiding av scriptene er dette fokusert på, slik at analytiske element overføres til korrekt posisjon.

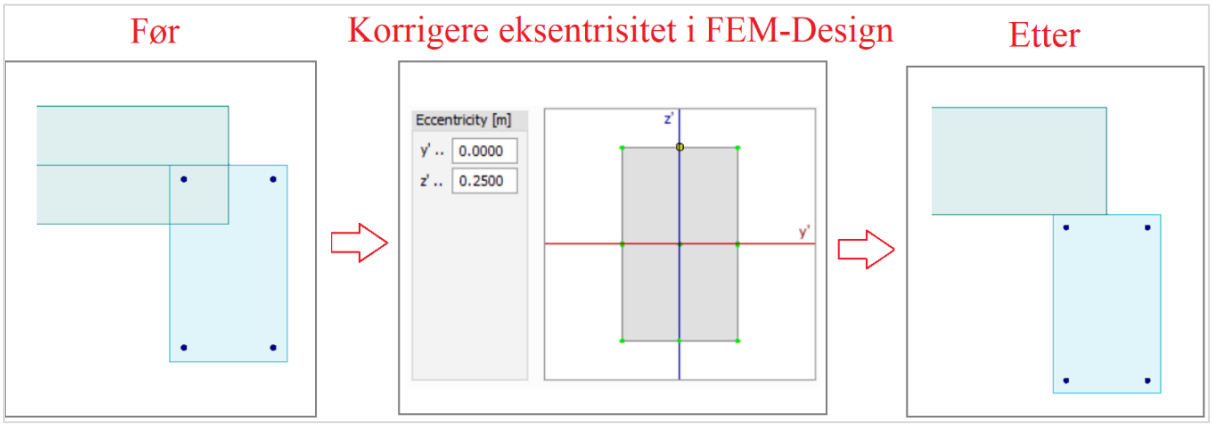
6.3.2.1 Armering i søyler og bjelker

Etter det er etablert en armeringmodell i FEM-Design er det forsøkt å overføre den til Revit, uten å korrigere eksentrisitet, med bruk av både script og StruXML. StruXML overfører den eksakte armeringsmodellen fra FEM-Design til Revit, uten å gjøre noen korreksjoner. Dette resulterer i at armering havner utenfor tilhørende element, som vist til venstre i *figur 6.6*. Scriptet overfører også den eksakte armeringsmodellen fra FEM-Design til Revit, men samtidig utfører en korreksjon av eksentrisitet til analytiske elementer. Dette gjør at armeringen havner innenfor tilhørende element i Revit, som vist til høyre i *figur 6.6.*



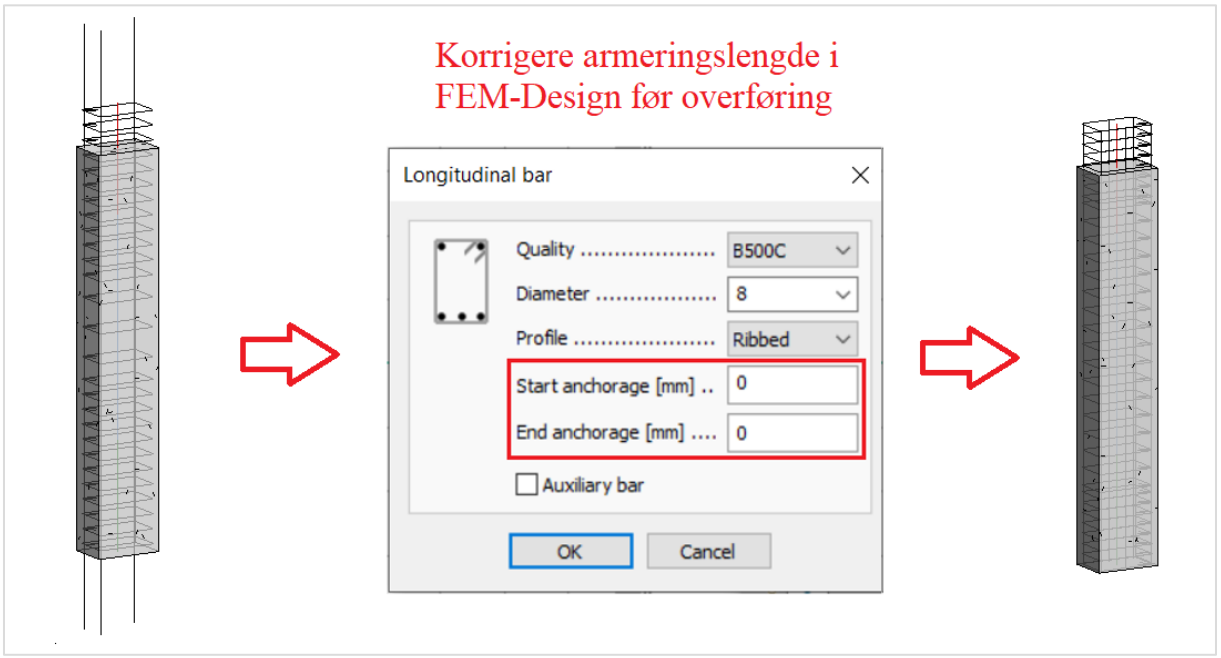
Figur 6.6 Viser overføring av armeringsmodell ved bruk av StruXML (Til venstre) og script (Til høyre).

Dersom bruker korrigerer armeringsmodellen i FEM-Design før overføring med StruXML, oppnås et bedre resultat. *Figur 6.7* viser hvordan eksentrisitet til en bjelke korrigeres i FEM-Design.



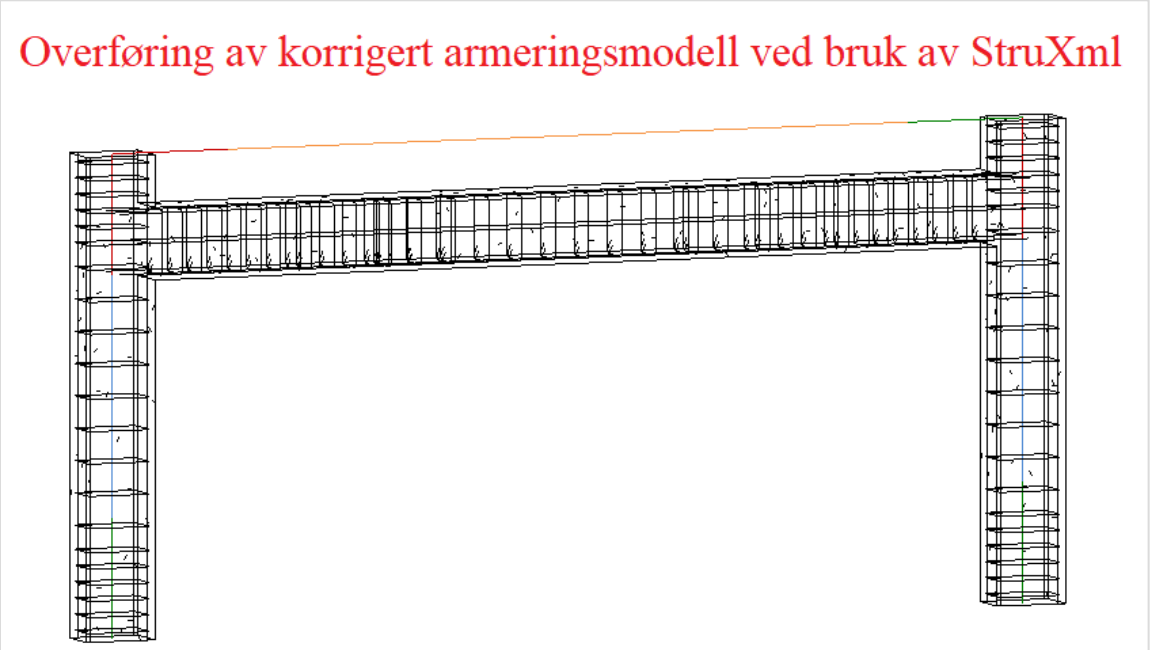
Figur 6.7: Viser hvordan eksentrisitet kan korrigeres i FEM-Design.

Lengdearmeringen i søyler og bjelker er plassert som single armeringsjern og inneholder en forankringslengde fra FEM-Design. Dersom lengdearmering skal avsluttes i elementet er det nødvendig å korrigere det i FEM-Design før overføring til Revit, vist i *figur 6.8*. Siden armering i FEM-Design baserer seg på analytiske linjer, er start- og sluttpunkt for analytiske linjer definert som nullpunkt. Dersom armering skal avsluttes før dekke, må også dekketykkelse og overdekning trekkes fra.



Figur 6.8 Viser hvordan lengdearmering kan reduseres i FEM-Design

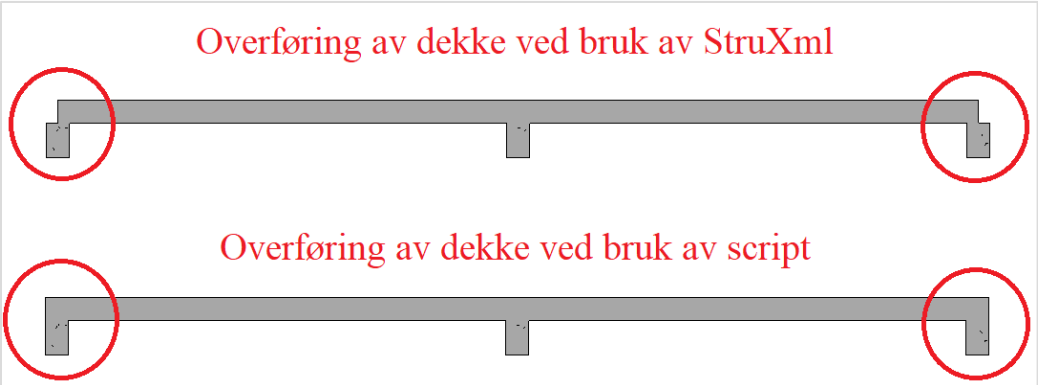
For bøylearmering er det ikke funnet en metode som reduserer lengden, slik at armering passer inn i elementet. Det er også forsøkt å redusere lengden til analytiske linjer, men uten hell. Om ikke det eksisterer en metode for å fikse dette, vil en korrigert armeringsmodell bli overført med StruXML som vist i *figur 6.9*



Figur 6.9 Viser hvordan armeringsmodell overføres til Revit med StruXML. Det er også nødvendig å redusere søylehøyde.

6.3.2.2 Armering i dekke

Som nevnt i *delkapittel 5.3.4* går ikke dekkearmering helt ut til dekkekant, men stopper med analytiske linjer. Dette er noe som er tatt hensyn til i scriptet, men må korrigeres i Revit/FEM-Design med bruk av StruXML, som vist i *figur 6.10.*



Figur 6.10 Viser hvordan dekke ikke går helt ut til dekkekant ved bruk av StruXML

Det er alltid behov for endebøyler i et dekke. En armeringsmodell fra FEM-Design inneholder ikke endebøyler og vil dermed ikke lages med StruXML sin overføring. Dette er noe som er implementert i scriptet og lages med bruk av noen få inputs.

6.3.2.3 Oppsumering av sammenligning mellom script og StruXML

Foreslått script er sammenlignet med StruXML sin overføring. Foreløpig kan scriptene overføre rektangulær geomtri til sammenling med StruXML som gir mer fleksibilitet. Dermed må script videreutvikles slik at det håndterer overføring av flere typer tverrsnitt. Scriptene trenger også ytterligere testing og verifisering på flere modeller for å øke påliteligheten til scriptet. Til sammenligning har StruXML vært kjent i bransjen en stund, og er testet for flere typer modeller.

7 Konklusjon

Målet for oppgaven var å utforske hvordan parametrisk design kan anvendes for å bidra til en mer effektiv prosjektering. Her var det et ønske å etablere en arbeidsflyt mellom tegningsprogrammet Revit og bereningsprogrammet FEM-Design. Ved bruk av et visuelt programmeringsverktøy, Dynamo, er det utarbeidet to forslag til script som ser på hvordan samhandlingen mellom programmene kan utnyttes for å bidra til en mer effektiv prosjektering.

- Første script tar for seg en overføring av analytisk modell fra Revit til FEM-Design. Det vil også etablere laster, lastkombinasjoner, etasjer, akser, opplagerbetingelser og forbindelse mellom elementer. Til slutt utfører scriptet en analyse som gir krefter på konstruksjonen i ULS og SLS. Dette oppnås med bruk av kun 6 inputs, som er lite tidkrevende å endre på. Hvordan krefter fordeler seg i en konstruksjon er svært variabelt, og er tilnærmet umulig å automatisere. Likevel viser scriptet at store deler av en beregningsmodell kan automatiseres, men noe manuelt arbeid må gjøres.

- Andre script overfører en armeringsmodell fra FEM-Design til Revit, samt gir brukeren et valg om å overføre strukturelle elementer. FEM-Design lager armering utifra den analytisk linjen til et element. Siden Revit jobber med en fysisk modell, er det ofte behov for å korrigere analytiske linjer, slik at armering overføres riktig til Revit. Dette er noe som er satt søkelys på ved utvikling av scriptet og resultatet viser at det håndteres på en god måte. Det er også inkludert en metode som lager endebøyler i dekke, noe som ikke en armeringsmodell fra FEM-Design inneholder. Ved bruk av kun 7 input vil scriptet lage en armeringsmodell i Revit og vurderes som en nøyaktig og effektiv løsning.

- Det eksisterer allerede i dag verktøy som utfører en tilsvarende overføring som scriptene. Dette verktøyet er StruXML som er add-inn i Revit og laget av Strusoft. StruXML er benyttet i oppgaven for at resultat til scriptene kan sammenlignes opp mot eksisterende verktøy. Til tross for at scriptene har sine begrensninger, eksisterer det noen fordeler som ikke StruXML tilbyr.

- Scriptene tar utgangspunkt i en forenklet modell, men er også testet på forskjellige modeller for å validere at overføringen fungerer som den skal. Det er noen forutsetninger for at en overføring skal være velykket, men disse er minimale sett opp mot manuelt arbeid. Resultatene anses å være nøyaktige og tidsbesparende. Med bakgrunn i dette, konkluderes det med at implementering av parametrisk design i en prosjekteringsfase bidrar til et mer effektivt resultat.

8 Videreutvikling av script

Som videreutvikling av scriptet er det naturlig å bygge videre på begrensninger som er definert i *delkapittel 6.2.1*. Under arbeidet med oppgaven er det oppdaget noen måter å forbedre scriptet på som kan resultere i et mer optimalt script. Siden oppgaven utforsker et veldig åpent og omfattende tema, er primære utbedringer grunnet avgrensninger av oppgaven og tidsmessige årsaker.

- Første utbedring er å tilrettelegge for at scriptet kan brukes for all type geometri, og ikke bare rektangulære tverrsnitt. Her er det også interessant å utforske om det kan lages et script for å håndtere mer kompleks geometri, noe som er begrenset i Revit og FEM-Design. Sannsynligvis vil det kreve noe arbeid i tekstbasert koding for å oppnå dette.
- Som nevnt i *delkapittel 6.2.1* vil det være mer tidkrevende å kjøre scriptet for større modeller. Dette er grunnet at det er mange operasjoner som kjøres samtidig. Siden det er utallig mange metoder å komme frem til samme resultat, er det sannsynlig at det eksisterer en mer optimal løsning. Dermed kan viderearbeid være å finne operasjoner i scriptet som kan forbedres og optimaliseres, noe som fører til et mer effektiv script.

9 Referanser

Allplan. (2020, April 24). Parametric BIM Modeling - Efficiency And Flexibility In Planning Processes.

Allplan. (2021, august 25). *The Business Benefits of 3D Rebar Detailing*. Retrieved from https://blog.allplan.com/en/the-business-benefits-of-3d-rebar-detailing

Arup. (n.d.). *Parametric design for better buildings*. Retrieved from Arup: https://www.arup.com/perspectives/parametric-design-for-better-buildings

Asti. (n.d.). *Ultimare guide to autodesk Dynamo*. Retrieved from https://www.asti.com/ultimate-guide-to-autodesk-dynamo/

Azevedo, V. (2014). Retrieved from https://www.semanticscholar.org/paper/BIM-model-analysis-on-a-structural-design-Azevedo/6186e58a962cdfea9334e0c25c9447ab625cf271

Bjergø, I. B. (2020, September 23). *Reinforcement*. Retrieved from https://wiki.fem-design.strusoft.com/xwiki/bin/view/BIM%20Integration/StruXML%20Revit%20%20Add-In/Import%20StruXml/Import%20scope/Reinforcement/

Bjergø, I. B. (2022, april 27).

Brito, J. P. (2019, August 18). *BIM AEC*. Retrieved from http://www.bim-aec.com/2019/08/18/dynamo-takes-bim-to-the-next-level/

Construsoft. (n.d.). *The Benefits of Parametric Design for Structural Engineers.* Retrieved from https://www.construsoft.com/news/benefits-parametric-design-structural-engineers

Cornelius. (2018, September 10). *Ultimate Limit States (ULS) and Serviceability Limit States (SLS)*. Retrieved from https://engineeringbasic.com/ultimate-limit-states-uls-and-serviceability-limit-states-sls/

Dynamo Primer. (2019). *The Dynamo Primer*. Retrieved from https://primer.dynamobim.org/index.html

Dynamo Primer. (2019). *The Dynamo Primer*. Retrieved from https://primer.dynamobim.org/08_Dynamo-for-Revit/8-2_Selecting.html

Easel. (n.d.). Retrieved from Easel: https://www.easel.ly/create?id=https://s3.amazonaws.com/easel.ly/all_easels/5854805/HowKetosisWork&key=pri#

Enweetech. (n.d.). Retrieved from https://enveetech.com/rebar-detailing-services

Gillis, A. S. (2022, 5). *Algorithm*. Retrieved from Techtarget: https://www.techtarget.com/whatis/definition/algorithm

Hamil, D. S. (2021, September 21). Dr Stephen Hamil.

Hejnfelt, A. (n.d.). *From Revit to FEM and back – with use of Dynamo.*

Indovance. (2022, Mars 1). *The FUTURE of Rebar Detailing in a Fast-paced Construction Industry.* Retrieved from Inodvance: https://www.indovance.com/knowledge-center/the-future-of-rebar-detailing-in-a-fast-paced-construction-industry/

Jalli, A. (2022). *What Is Programming?* Retrieved from Codingem: https://www.codingem.com/what-is-programming/

Kilkelly, M. (2018, Februar 8). *What Is Dynamo and 5 Reasons You Should be Using It*. Retrieved from https://archsmarter.com/what-is-dynamo-revit/

Knittle, B. (n.d.). *Content in Revit - Families.*

Lindholm, S. (n.d.). *The Benefits of Parametric Design for Structural Engineers*. Retrieved from Tekla: https://www.tekla.com/resources/blogs/the-benefits-of-parametric-design

Midttun, F. (2018, 11 12). *Konkurransekraft og lønnsomhet i bygg, anlegg og næringseiendom*. Retrieved from BDO Norge: https://www.bdo.no/nb-no/bloggen/konkurransekraft-og-l%C3%B8nnsomhet-i-bygg,-anlegg-og-n%C3%A6ringseiendom

Molinos, R. (2016, September 19). *Data hierarchy.* Retrieved from Modelical: https://www.modelical.com/en/gdocs/revit-data-hierarchy/

NordicBIM. (n.d.). *Building information modelling*. Retrieved from https://www.nordicbim.com/no/alt-om-bim-bygningsinformasjonsmodellering-fra-vugge-til-grav

Norsk betongforening. (2018, Oktober 26). Armering - Prosjektering og utførelse. p. 94. Retrieved from https://betong.net/wp-content/uploads/H%C3%B8ringsutkast-Publikasjon-nr.-8-2018-10-26.pdf

nti. (n.d.). *Autodesk Revit*. Retrieved from https://www.nti.biz/no/nti-catalog/software/autodesk-revit/

Rajput, K. (n.d.). *Live Load Vs Dead Load | What Is Load in Civil*. Retrieved from https://civiljungle.com/live-load-vs-dead-load/

Smith, C. (2020, Juli 30). *newcivilengineering.* Retrieved from Engineers missing out on the benefits of parametric design: https://www.newcivilengineer.com/latest/engineers-missing-out-on-the-benefits-of-parametric-design-30-07-2020/

Strusoft. (n.d.). *FEM-Design Wiki*. Retrieved from https://wiki.fem-design.strusoft.com/xwiki/bin/view/Main/

Tekla. (n.d.). The Six Most Common Challenges in Rebar Detailing and How to Overcome Them. Retrieved from https://www.tekla.com/resources/blogs/the-six-most-common-challenges-in-rebar-detailing-and-how-to-overcome-them

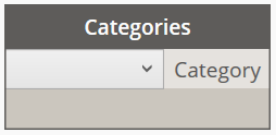
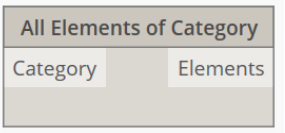
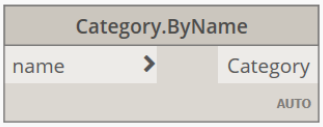
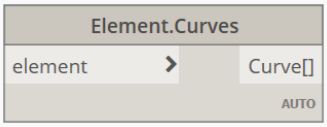
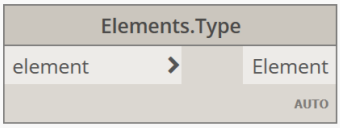
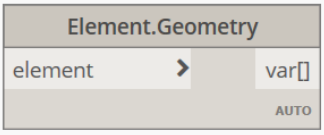
What is BIM (Building Information Modeling). (2022, 4 6). Retrieved from https://constructible.trimble.com/construction-industry/what-is-bim-building-information-modeling

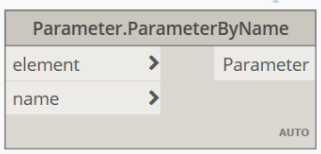
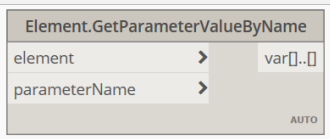
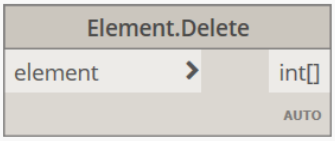
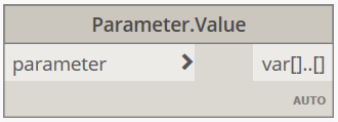
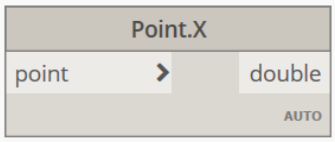
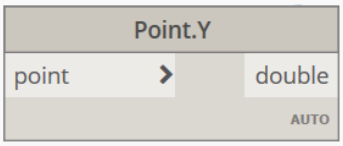
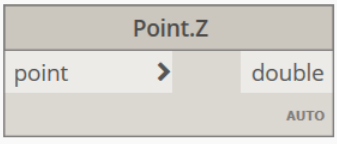
Wyatt, M. (n.d.). *What is an API? A Digestible Definition with API Examples for Ecommerce Owners*. Retrieved from Bigcommerce: https://www.bigcommerce.com/blog/what-is-an-api/#why-use-a-custom-api-solution

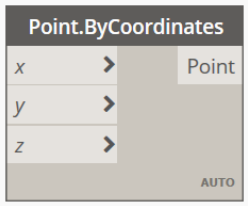
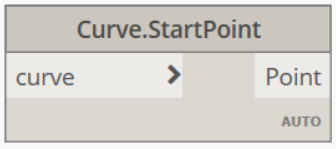
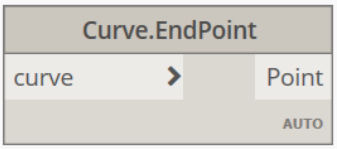
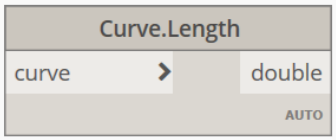
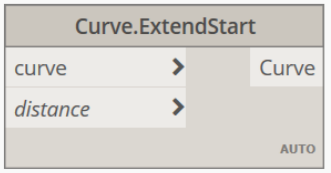
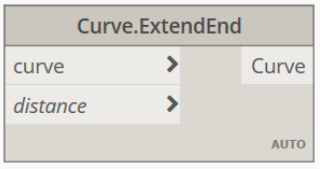
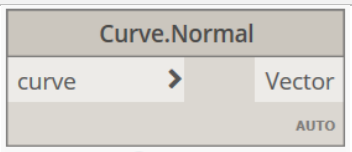
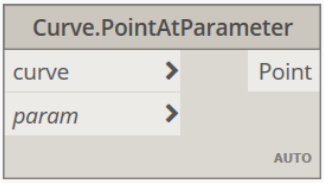
Aarhus, C. (2019, Juni 21). Podkast: Hva er vitsen med papirløse byggeplasser?

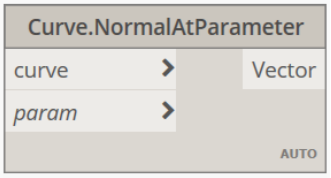
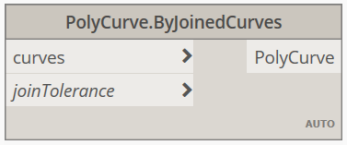
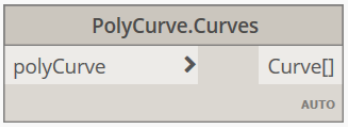
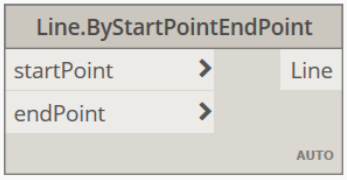
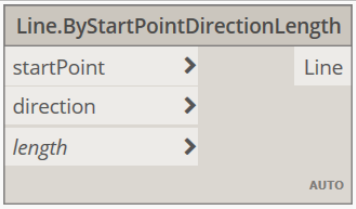
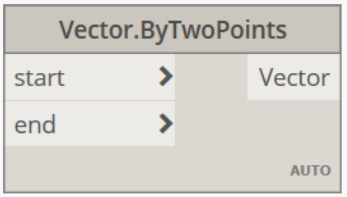
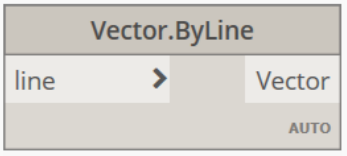
10 Vedlegg

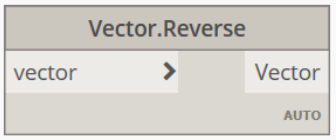
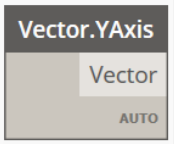
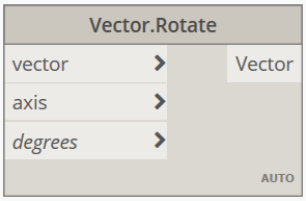
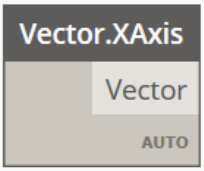
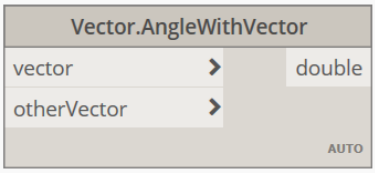
10.1 Oversikt over noder som er brukt i script

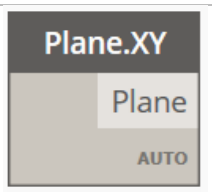
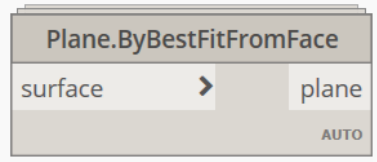
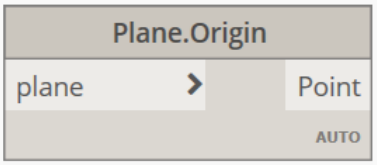
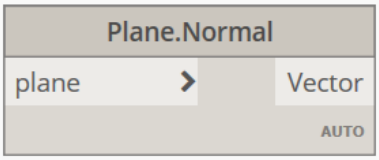
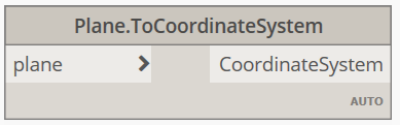
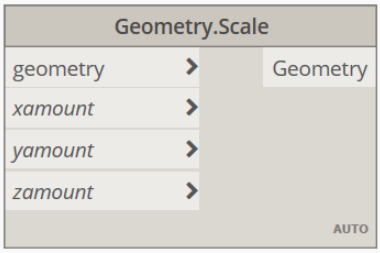
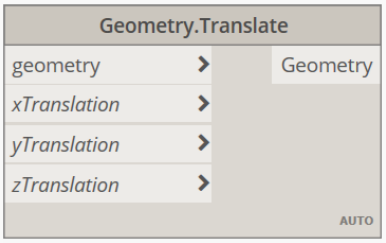
Code Block - Pakke: Script - En fleksibel node som brukes for å lage formler og koder	
Categories - Pakke: Revit - Alle kategorier i en Revit modell	
All Elements of category - Pakke: Revit - Hente alle elementer til den spesifiserte kategori	
Category.ByName - Pakke: Revit.Selection - Henter en kategori basert på kategorien sitt navn i en Revit modell	
Element.Curves - Pakke: Revit - Henter alle kurver til et element	
Elements.Type - Archilab - Hente element type fra et element	
Element.Geometry - Pakke: Revit - Henter all tilhørende geometri til et element	

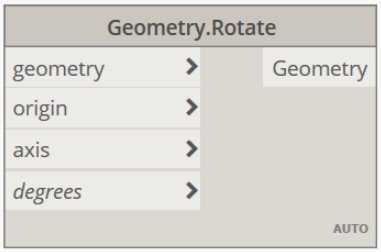
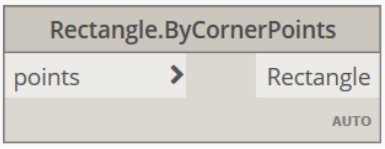
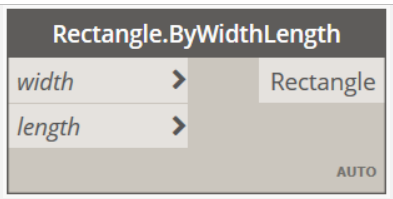
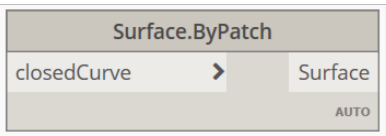
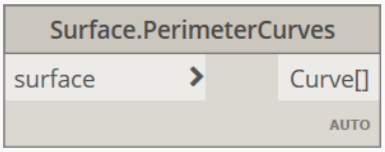
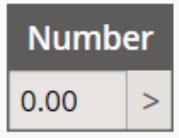
Parameter.ParameterByName - Pakke: Revit - Henter en parameter til et element basert på parameternavn	
Element.GetParameterValueByName - Pakke: Revit - Henter en parameterverdi basert på parameternavnet	
Element.Delete - Pakke: Revit - Sletter element fra Revit modell	
Parameter.Value - Pakke: Revit - Henter en parameter verdi	
Point.X - Pakke: Geometry - X koordinat til et punkt	
Point.Y - Pakke: Geometry - Y koordinat til et punkt	
Point.Z - Pakke: Geometry - Z koordinat til et punkt	

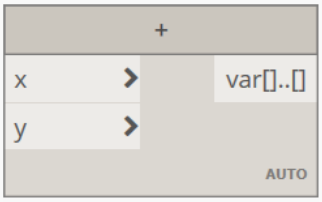
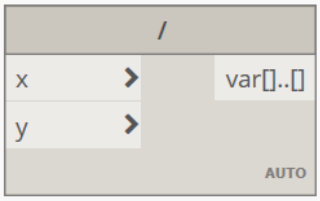
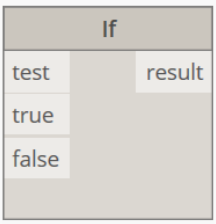
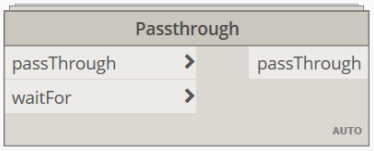
Point.ByCoordinates - Pakke: Geometry - Lager et punkt basert på X,Y og Z koordinater	
Curve.StartPoint - Pakke: Geometry - Gir startpunktet til en kurve	
Curve.EndPoint - Pakke: Geometry - Gir sluttpunktet til en kurve	
Curve.Length - Pakke: Geometry - Gir lengden til en kurve	
Curve.ExtendStart - Pakke: Geometry - Forlenger/krymper en kurve fra startpunkt	
Curve.ExtendEnd - Pakke: Geometry - Forlenger/krymper en kurve fra sluttpunkt	
Curve.Normal - Pakke: Geometry - Lager en normalvektor på en kurve	
Curve.PointAtParameter - Pakke: Geometry - Lager et punkt på en kurve basert på en faktor fra 0 til 1	

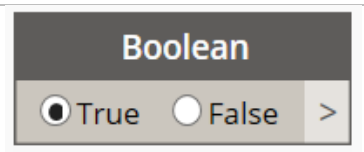
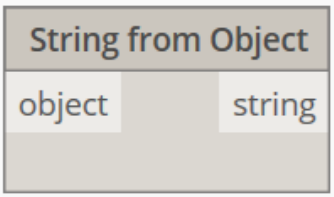
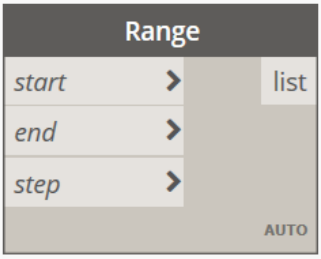
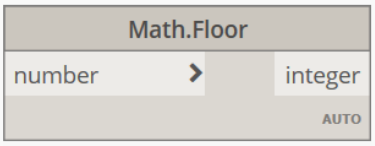
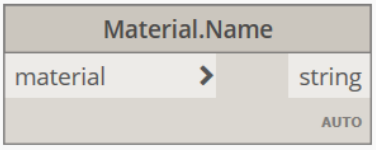
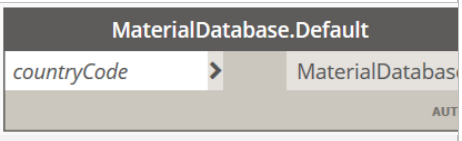
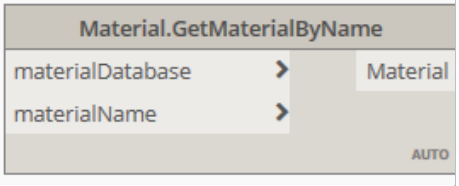
Curve.NormalAtParameter - Pakke: Geometry - Gir en vektor som går normalt på en kurve i plan	
PolyCurve.ByJoinedCurves - Pakke: Geometry - Samler og lukker kurver	
PolyCurve.Curves - Pakke: Geometry - Gjør Polycurver til kurver	
Line.ByStartPointEndPoint - Pakke: Geometry - Lager en linje basert på et startpunkt og sluttpunkt	
Line.ByStartPointDirectionLength - Pakke: Geometry - Lager en linje basert på et startpunkt, en vektor og en lengde	
Vector.ByTwoPoints - Pakke: Geometry - Lager en vektor basert på et startpunkt og sluttpunkt	
Vector.ByLine - Pakke: Geometry - Lager en vektor basert på en linje	

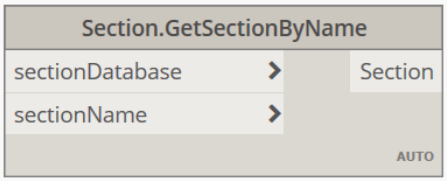
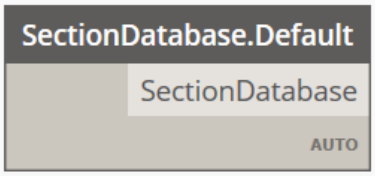
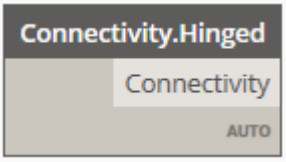
Vector.Z - Pakke: Geometry - Vektor i Z retning	
Vector.Reverse - Pakke: Geometry - Gir en vektor i motsatt retning enn den orginale retning	
Vector.Yaxis - Pakke: Geometry - En vektor langs Y-akse	
Vector.Rotate - Pakke: Geometry - Roterer en vektor med en vinkel utifra en akse	
Vector.IsParallell - Pakke: Geometry - Sjekker om en vektor er parallell med en annen vektor. Gir resultat som true eller false	
Vector.XAxis - Pakke: Geometry - En vektor langs X aksen	
Vector.AngleWithVector - Pakke: Geometry - Finner vinkelen mellom to vektorer	

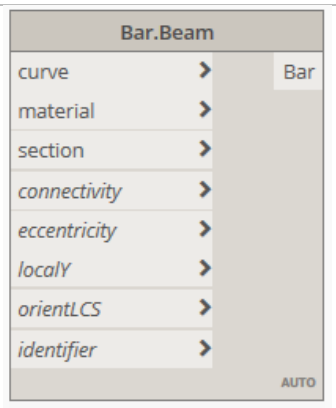
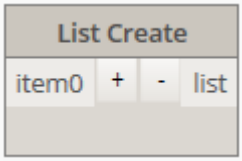
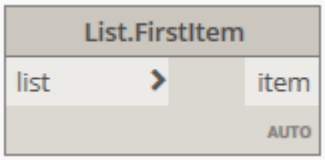
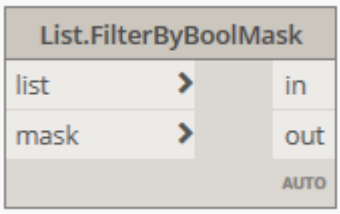
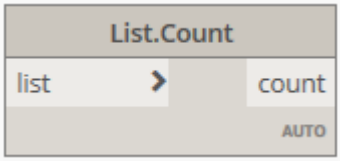
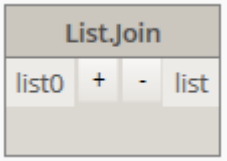
Plane.XY - Pakke: Geometry - Gir et XY plan	
Plane.ByBestFitFromFace - Pakke: Clockwork - Gir et plan basert på en overflate	
Plane.Origin - Pakke: Geometry - Gir midtpunktet til et plan	
Plane.Normal - Pakke: Geometry - Gir normalvektoren til et plan	
Plane.ToCoordinateSystem - Pakke: Geometry - Gir koordinatsystemet til et plan	
Geometry.Scale - Pakke: Geometry - Skalerer geometri med et forholdstall	
Geometry.Translate - Pakke: Geometry - Flytter geometri i ønsket retning basert på verdien som bruker velger	

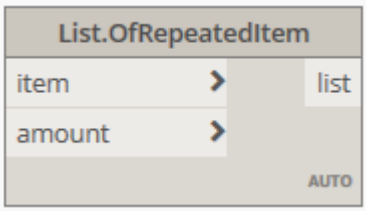
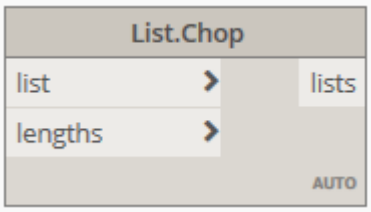
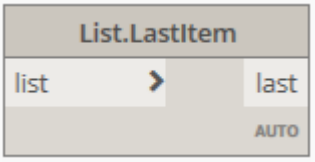
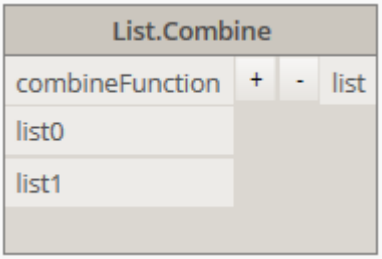
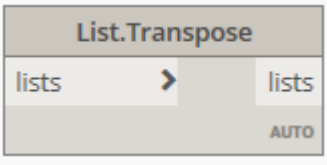
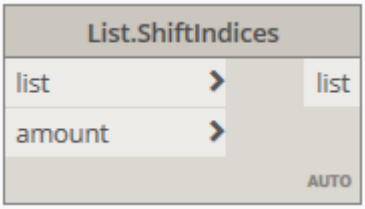
Geometry.Rotate - Pakke: Geometry - Roterer geometri med en vinkel rundt en akse, med utganspunkt i et punkt	
Rectangle.ByCornerPoints - Pakke: Geometry - Lager et rektangel basert på hjørnepunkt	
Rectangle.ByWidthLength - Pakke: Geometry - Lager et rektangel basert på en lengde og bredde	
Surface.Offset - Pakke: Geometry - Flytter en overflate med en bestemt avstand	
Surface.ByPatch - Pakke: Geometry - Lager en overflate basert på en lukket kurve	
Surface.PerimeterCurves - Pakke: Geometry - Gir kantlinjer til en overflate	
Number - Pakke Input - Brukes for tall som skal være input for et script	

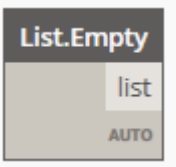
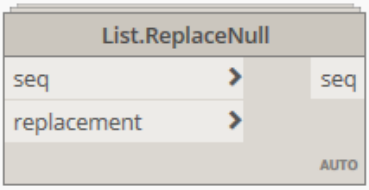
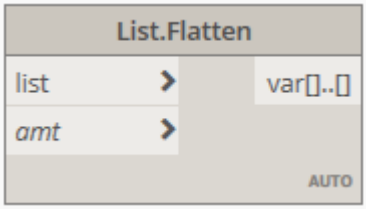
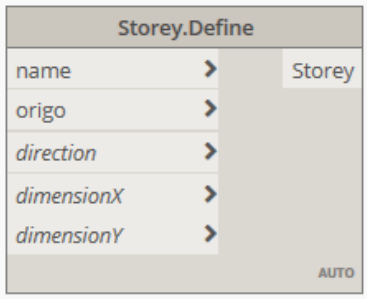
== - Pakke: Math - Sjekker om X er lik som Y. Resultatet gis som true eller false	
+ - Pakke: Math - Brukes for å addere X med Y. Brukes oftest Code Block for denne operasjonen, men dersom det er behov for å velge et nivå i en liste må denne brukes.	
- - Pakke: Math - Brukes for å subtrahere X med Y	
/ - Pakke: Math - Brukes for å dividere X med Y	
If - Pakke: Script - Gir et resulatet basert på om et utsagn er true eller false.	
PassThrough - Pakke: Clockwork - Venter at en prosess er utført før en annen prosess utføres	

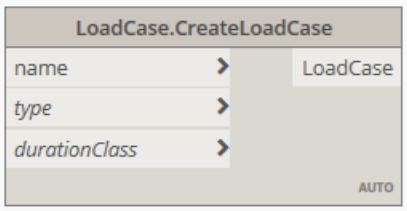
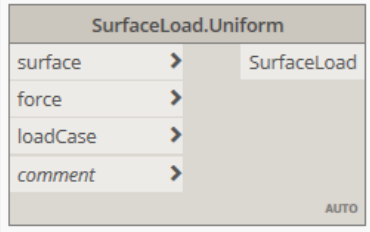
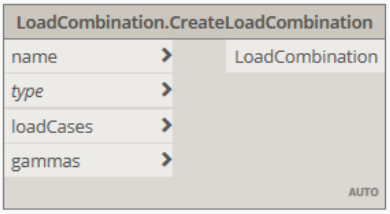
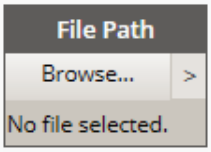
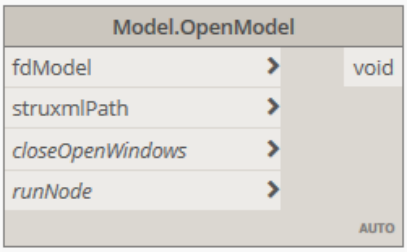
Boolean - Pakke: Input - Node som gir brukeren mulighet til å velge mellom en test er true eller false	
String from Object - Pakke: String - Gjør et objekt navn om til tekst.	
Range - Pakke: List - Lager en liste basert på et start- og sluttnummer med et definert stigningstall	
Math.Floor - Pakke: Math - Runder ned til nærmeste hele tall	
Material.Name - Pakke: Revit - Gir navnet til en material type i Revit	
MaterialDatabase.Default - Pakke: FEM-Design - Henter alle materialer fra FEM-Design sin database	
Material.GetMaterialByName - Pakke: FEM-Design - Gir en material type basert på en database og et materialnavn	

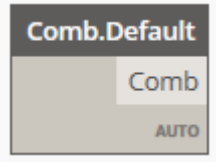
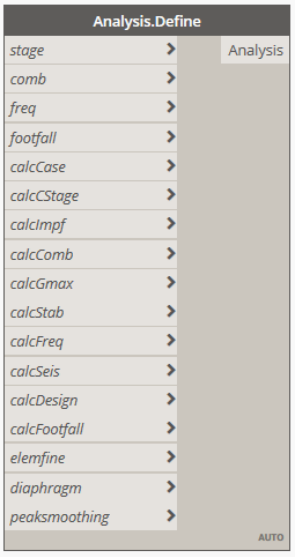
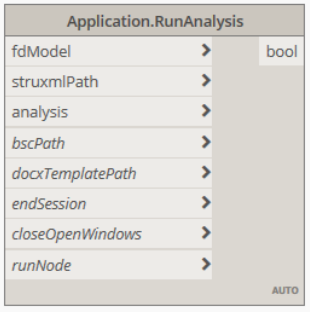
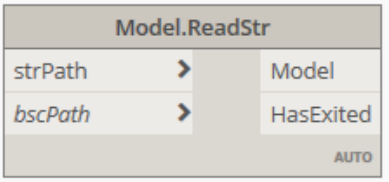
Section.GetSectionByName - Pakke: FEM-Design - Gir et tverrsnitt basert på en database og et tverssnitt navn	
SectionDatabase.Default - Pakke: FEM-Design - Henter alle tverrsnitt fra FEM-Design sin database	
Connectivity.Rigid - Pakke: FEM-Design - Bestemmer at en forbindelse mellom elementer skal være stiv	
Connectivity.Hinged - Pakke: FEM-Design - Bestemmer at en forbindelse mellom elementer skal være leddet	
Bar.Column - Pakke: FEM-Design - Lager en søyle i FEM-Design	

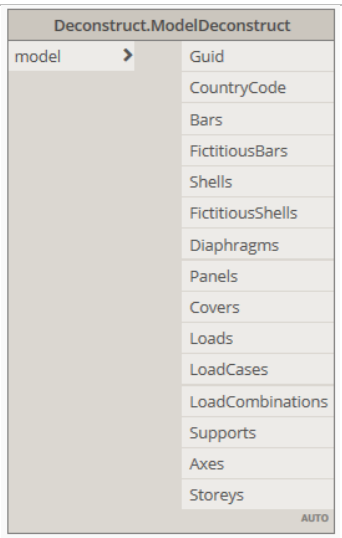
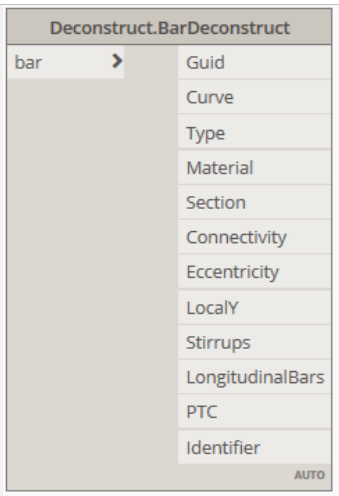
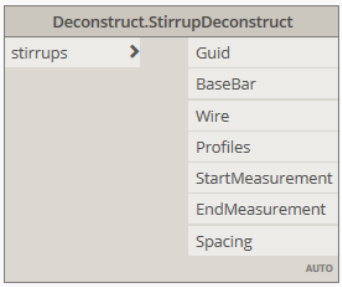
Bar.Beam - Pakke: FEM-Design - Lager en bjelke i FEM-Design	
List.Create - Pakke: List - Lager en liste fra andre lister/items.	
List.FirstItem - Pakke: List - Henter det første punktet fra en liste	
List.MinimumItem - Pakke: List - Henter punktet med lavest verdi	
List.FilterByBoolMask - Pakke: List - Filtrerer en liste basert på en betingelse	
List.Count - Pakke: List - Gir antall punkt det er i en liste	
List.Join - Pakke: List - Lager en ny liste basert på tilkoblede lister/punkt	

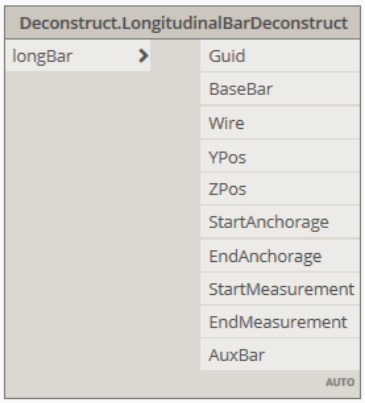
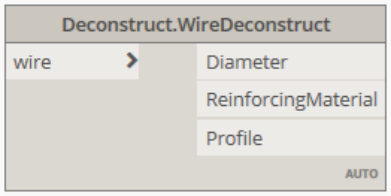
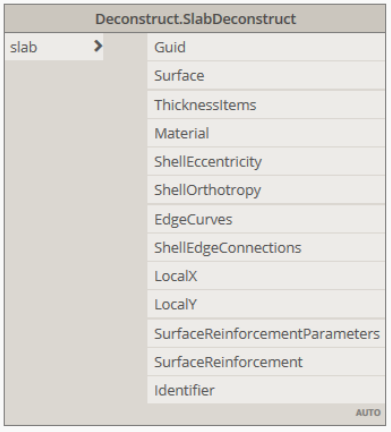
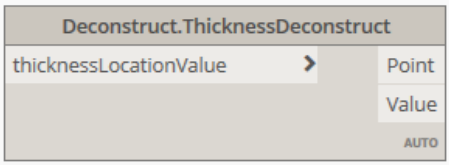
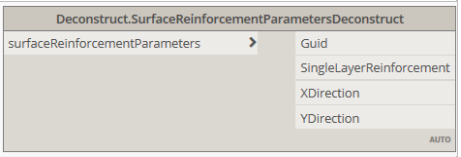
List.OfRepeatedItem - Pakke: List - Repeterer en liste antall ganger som angitt	
List.Chop - Pakke: List - Deler opp enn liste til mindre underlister basert på hvor mange punkt som angis for hver underliste	
List.SortByKey - Pakke: List - Sorterer en liste basert på et utsagn	
List.LastItem - Pakke: List - Henter siste punkt i en liste	
List.Combine - Pakke: List - Kombinerer lister basert på en funksjon	
List.Transpose - Pakke: List - Brukes for å bytte rader med kolonner ved flere lister i en liste	
List.ShiftIndices - Pakke: List - Brukes for å forskyve en indeks til høyre i listen.	

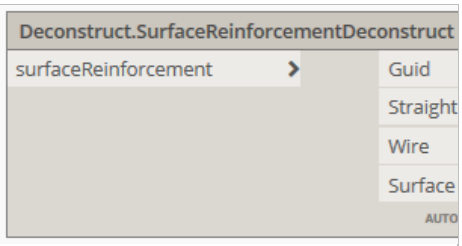
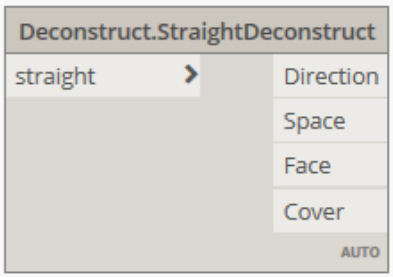
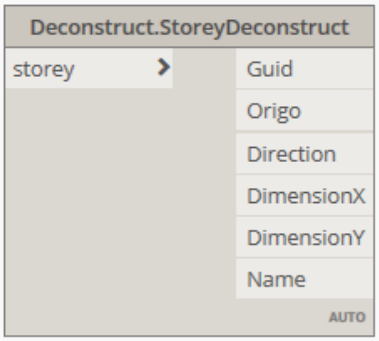
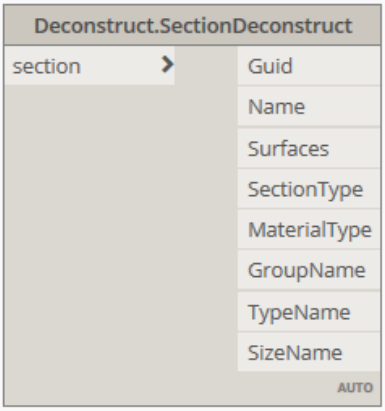
List.Empty - Pakke: List - Gir en tom liste	
List.AddItemToFront - Pakke: List - Legger til et punkt først i en liste	
List.ReplaceNull - Pakke: List - Skifter ut nullverdier i en liste med en verdi/tekst	
List.Flatten - Pakke: List - Reduserer antall levels i en liste med en angitt mengde	
Axis.Define - Pakke: FEM-Design - Lager akser i FEM-Design	
Storey.Define - Pakke: FEM-Design - Lager etasjer i FEM-Design	

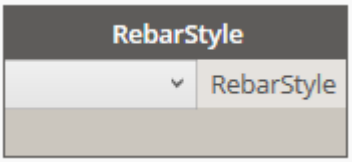
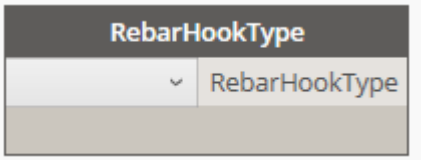
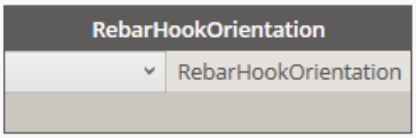
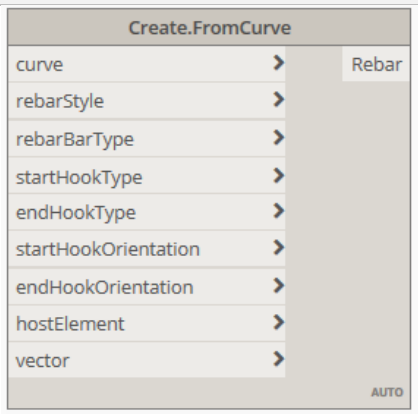
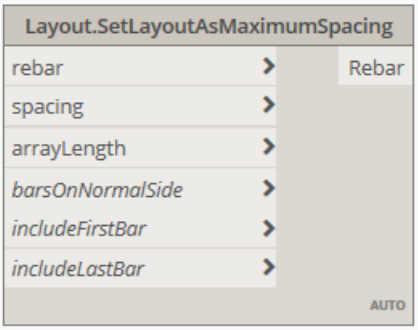
LoadCase.CreateLoadCase - Pakke: FEM-Design - Lager en lasttilfelle i FEM-Design	
SurfaceLoad.Uniform - Pakke: FEM-Design - Lager en overflate last i FEM-Design	
LoadCombination.CreateLoadCombination - Pakke: FEM-Design - Lager en lastkombinasjon i FEM-Design	
Model.CreateNewModel - Pakke: FEM-Design - Lager en FEM-Design modell	
File Path - Pakke: FEM-Design - Brukes for å velge en fil	
Model.OpenModel - Pakke: FEM-Design - Åpner en FEM-Design modell	

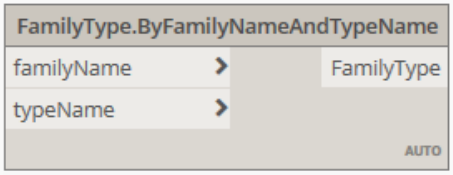
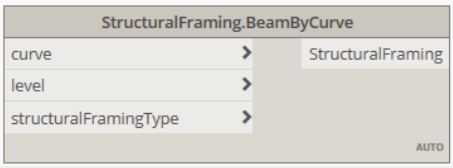
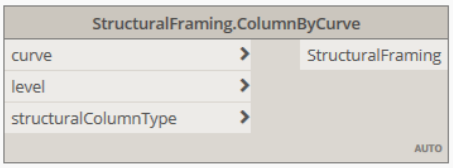
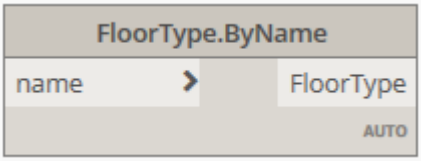
Comb.Default - Pakke: FEM-Design - Brukes for å velge hvilken analyse som skal utføres	
Analysis.Define - Pakke: FEM-Design - Brukes for å velge hvilken analyse som skal utføres	
Application.RunAnalysis - Pakke: FEM-Design - Brukes for å kjøre en analyse i FEM-Design	
Model.ReadStr - Pakke: FEM-Design - Leser en FEM-Design modell og henter all informasjon rundt modellen	

Deconstruct.ModelDeconstruct - Pakke: FEM-Design - Bryter ned en modell til mindre kategorier som videre kan brytest opp	
Deconstruct.BarDeconstruct - Pakke: FEM-Design - Bryter ned et bar element for å gi underliggende informasjon	
Deconstruct.StirrupDeconstruct - Pakke: FEM-Design - Bryter ned bøyler for å gi underliggende informasjon	

Deconstruct.LongitudinalBarDeconstruct - Pakke: FEM-Design - Bryter ned lengdearmering for å gi underliggende informasjon	
Deconstruct.WireDeconstruct - Pakke: FEM-Design - Bryter ned et armeringsjern for å gi underliggende informasjon	
Deconstruct.SlabDeconstruct - Pakke: FEM-Design - Bryter ned et dekke for å gi underliggende informasjon	
Deconstruct.ThicknessDeconstruct - Pakke: FEM-Design - Gir tykkelsen til et dekke	
Deconstruct. **SurfaceReinforcementParametersDeconstruct** - Pakke: FEM-Design - Bryter ned overflate armering for å gi underliggende informasjon	

Deconstruct.SurfaceReinforcementDeconstruct - Pakke: FEM-Design - Bryter ned overflate armering for å gi underliggende informasjon	
Deconstruct.StraightDeconstruct - Pakke: FEM-Design - Bryter ned «straigth» for å gi underliggende informasjon	
Deconstruct.StoreyDeconstruct - Pakke: FEM-Design - Bryter ned etasjer for å gi underliggende informasjon	
Deconstruct.SectionDeconstruct - Pakke: FEM-Design - Bryter ned et tverrsnitt for å gi underliggende informasjon	
Properties.RebarDiameter - Pakke: Structural Design - Gir diameter verdi til et an armeringstype	

RebarStyle - Pakke: Structural Design - Gir et valg mellom lengdearmering eller bøylearmering	
RebarHookType - Pakke: Structural Design - Gir et valg om hvordan bøylearmering skal avsluttes	
RebarHookOrientation - Pakke: Structural Design - Gir et valg om avsluttningen på bøylearmering skal gå til venstre eller til høyre	
Create.FromCurve - Pakke: Structural Design - Lager armering i Revit	
Layout.SetLayoutAsMaximumSpacing - Pakke: Structural Design - Brukes for å samle armeringsjern i grupper	

FamilyType.ByFamilyByNameAndTypeName - Pakke: Revit - Lager en familie type basert på et familienavn og et typenavn	
StructuralFraming.BeamByCurve - Pakke: Revit - Lager en bjelke i Revit	
StructuralFraming.ColumnByCurve - Pakke: Revit - Lager en søyle i Revit	
FloorType.ByName - Pakke: Revit - Gir en gulvtype basert på et navn.	
Floor.ByOutlineTypeAndLevel - Pakke: Revit - Lager et dekke i Revit