

I det følgende beskrives hvordan mindre og mellemstore danske udviklingsvirksomheder, der ikke har en etableret og objektorienteret designproces og tilhørende værktøjer, kan anvende Unified Modeling Language (UML) til design af embedded (indlejrede) softwaresystemer på en omkostningsbevidst måde.

Mange virksomheder ønsker i disse år at indføre bedre metoder til design og dokumentation af deres udviklingsprojekter, og kigger ofte på Unified Modeling Language (UML) som en måde at opnå dette på. En virksomhed kan have forskellige motiver for at gøre dette. Nogle af de mest almindelige motiver er:

- Forbedring af kvaliteten på de udviklede produkter. Virksomheden har typisk tidligere oplevet en række problemer med tidligere udviklede produkter, såsom store tidsforsinkelser, kundeklager over dårlig kvalitet, m.m.
- Hurtigere omstilling af udviklingsaktiviteterne når markedets behov for produkterne ændrer sig.
- Reduktion af udviklingstiden for nye produkter, gennem genbrug af dele af kode og design fra tidligere projekter.
- Et ønske om at sikre, at nye medarbejdere kan overtage og videreføre kildekoden.

## UML diagram oversigt

UML er en officiel standard, som kontrolleres af The Object Management Group (OMG - [www.omg.org](http://www.omg.org)). OMG har gennem tiden udgivet en række forskellige versioner af UML. Disse er opdelt i to hoved-grupper, UML 1 og UML 2. Den seneste version i UML 1 serien er version 1.5, og i UML 2 serien er det version 2.0. Visse simple UML-værktøjer supporterer dog kun UML v1, hvilket imidlertid er tilstrækkeligt for den anvendelse som beskrives i denne artikel. Men kan man vælge, bør man gå efter version 2.0 support.

UML 1 indeholder 10 forskellige diagramtyper, hvilket er øget til 13 i UML 2.:

- **Statiske (strukturelle) diagrammer:**
  - Class diagram
  - Object diagram
  - Package diagram
  - Component diagram
  - Deployment diagram
  - Composite structure diagram (kun i UML 2)
- **Dynamiske diagrammer:**
  - Use Case diagram
  - Sequence diagram
  - Communication diagram (kaldes Collaboration diagram i UML 1)
  - Activity diagram
  - State Machine diagram
  - Interaction overview diagram (kun i UML 2)
  - Timing diagram (kun i UML 2)

## Valg af detaljeringsniveau

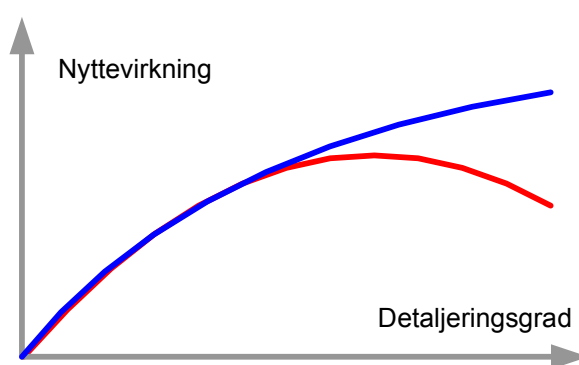
Umiddelbart vil UML-novicen forvente at skulle modellere det komplette softwaredesign i UML – det vil sige samtlige klasser, metoder, attributter, typer, state maskiner, m.m. Dette er imidlertid ikke noget absolut krav.

Der findes to hoved-skoler indenfor UML modellering: Den ene udgøres af de såkaldte 'kompletister', der går efter det komplette UML-design, mens den anden omfatter folk der mener at UML bruges bedst som et skitse-værktøj til at beskrive de vigtigste aspekter af designet.

'Kompletist'-skolen står bla. bag Model Driven Architecture (MDA), som er et forsøg på at bruge UML som et decideret programmeringssprog. Dette perspektiv er ganske tillokkende på lang sigt, men stiller også ganske store krav til både værktøjer og deltagerens faglige kunnen.

Den ønskede sammenhæng mellem detaljeringsgrad og nyttevirkning er vist med den blå kurve på Figur 1. Kurven flader ud for højere detaljeringsgrader, da tilføjelsen af spidsfindige detaljer til et stort set komplet design ikke kan forventes at have samme nyttevirkning som tilføjelsen af helt basal information i starten.

Førstegangsbrugeren risikerer dog nemt at ende på den røde kurve i stedet for, hvor tilføjelsen af flere detaljer ligefrem kan have en negativ virkning. Dette skyldes at spildtiden ved at 'fedte rundt' med model-detajler ikke altid opvejes af den designmæssige gevinst.



Figur 1 - Nyttvirkning af detaljeringsgraden

Vil man benytte sit UML-værktøj til at generere kode, er modellen selvsagt nødt til at have en eller anden grad af kompleksitet over sig. Men jeg vil råde førstegangsbrugeren til ikke at stræbe efter dette. Man vil sandsynligvis have nok at gøre med at omstille sig mentalt fra at have et tekst- og kode-baseret designopfattelse til en model-baseret ditto.

Brug derfor UML som skitseværktøj, og lad fortsat den traditionelle tekstbaserede specifikation være hjørnesteinen i designet. Den komplette model med MDA, kodegenerering, fløjter og guirlander kan passende indføres efter at man er blevet fortrolig med den basale UML anvendelse i løbet af de første to-tre projekter.

## Diagram-anvendelse

Der findes ingen faste regler for hvordan og til hvad man benytter de forskellige UML-diagrammer. De følgende anbefalinger er baseret på mine egne holdninger og erfaringer, men det enkelte udviklingsteam kan og bør tilpasse anvendelsen efter omstændighederne.

**Kravspecifikationen** bør beskrives ved hjælp af Use Cases på tekstuel form i et almindeligt (Word) dokument. Jeg finder personligt ikke at UML's Use Case diagrammer er synderligt værdifulde i forhold til den tid de tager at tegne og vedligeholde og den plads, de optager i kravdokumentet. Min anbefaling er ikke at spille tid og papir på dem, men i stedet for at benytte en fast skabelon for Use Case beskrivelserne. Eksempler på Use Case skabeloner kan hentes her: <http://alistair.cockburn.us/usecases/usecases.html>. Det skal dog nævnes, at flere UML

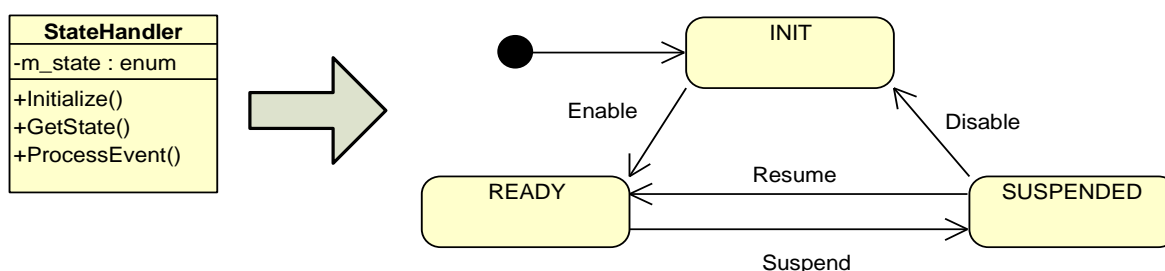
værktøjer efterhånden mulighed for at knytte (HTML-)formateret tekst til et Use Case diagram, hvilket evt. kan benyttes som alternativ til et eksternt dokument. Vil man absolut anvende Use Case diagrammer, gør man sig selv en stor tjeneste ved ikke at overdrive anvendelsen af relationer såsom <<uses>> og <<extends>> mellem Use Cases.

De **vigtigste Use Cases** bør suppleres med Sequence diagrammer, specielt når dynamisk opførsel skal beskrives. Sequence diagrammer giver dog ikke de store muligheder for at afbilde alternative scenarier, og derfor kan forekomme nødvendigt at fremstille et større antal diagrammer med ganske små variationer for at håndtere disse. Her er det ligeledes vigtigt at begrænse anvendelsen til de situationer, hvor diagrammerne virkelig gør nytte. Hvis to linier tekst i selve Use Case beskrivelsen kan gøre det ud for et ekstra sequence diagram med en lille ændring i forhold til hoved-diagrammet, er det helt acceptabelt at droppe det ekstra diagram.

**Systemets overordnede struktur** beskrives bedst i et antal Component diagrammer. Et alternativ er at benytte et Package diagram, men denne diagramtype er lidt for begrænset for min smag. Med et Component diagram har man mulighed for at beskrive vigtige interfaces mellem komponenterne. Hvis det er vigtigt at beskrive et messageflow mellem komponenter, kan man supplere med et Communication diagram (UML 1: collaboration diagram).

En enkelt **komponent** beskrives typisk i et Class diagram. Hovedvægten bør som ovenfor nævnt lægges på de store linier, herunder public metoder og interfaces. Interne (private) attributter og metoder er ikke voldsomt interessante. Et andet aspekt som typisk overdrives i et klasse-diagram er nedarvning. Nedarvning er typisk et af de første emner der behandles i enhver lærebog om objekt-orienteret design, men man skal her huske på at et klasse-diagram hører til den statiske beskrivelse af systemet, og for langt de fleste indlejrede systemer er de dynamiske egenskaber de vigtigste.

Såfremt en komponent eller klasse indeholder væsentlige **state maskiner**, beskrives disse i et State Machine diagram, som indikeret i Figur 2. Man bør dog ikke forsøge at vise samtlige kombinationer af tilstande og hændelser i dette diagram, da det ellers hurtigt bliver meget uoverskueligt. State diagrammet skal give et overblik over den *primære hensigt* med state maskinen, og bør derfor næsten altid suppleres med en **state-transition tabel**, som beskriver samtlige kombinationer af tilstande og hændelser, herunder alle mulige fejlscenarier. Denne tabel-type er ikke understøttet af UML, og må derfor beskrives i et separat dokument. Et eksempel er vist i Figur 3.



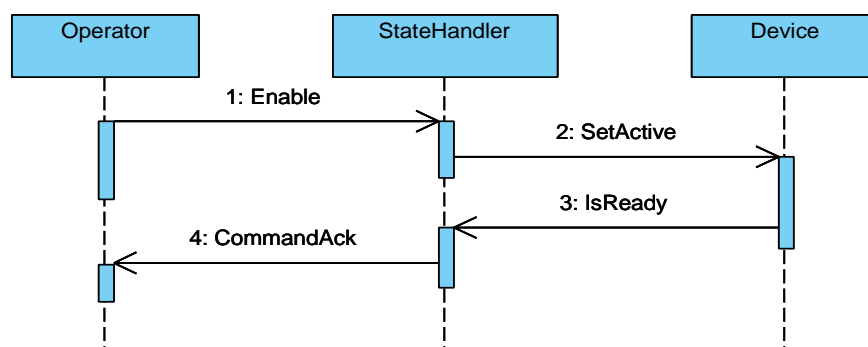
Figur 2 - Typisk state machine diagram

**Sekventielle processer**, såsom bruger-interaktion via en skærm eller en trinvis validering af modtagne data, beskrives bedst i et Activity diagram. Disse diagrammer minder meget om traditionelle flowchart diagrammer. I UML 2 er der blevet tilføjet elementer som minder om SDL modellering (kendt fra tele-verdenen til beskrivelse af kommunikationsprotokoller), og UML varianten kan anvendes til samme formål.

States	INIT	READY	SUSPENDED
Events			
<b>Enable</b>	Turn on stuff; READY	<ignore>	<ignore>
<b>Disable</b>	<error>	<error>	Turn off stuff; INIT
<b>Resume</b>	<error>	<ignore>	Enable sensor; READY
<b>Suspend</b>	<error>	Disable sensor; SUSPENDED	<ignore>

Figur 3 - State-transition tabel

**Asynkrone sekvenser**, såsom message-baserede protokoller kan også med fordel beskrives i et Sequence diagram, som indikeret i Figur 4. Dette diagram finder således både anvendelse til beskrivelse af krav såvel som design. Anvendelsen af diagrammerne bør ske efter de samme retningslinier som beskrevet ovenfor, nemlig at de bør beskrive de vigtigste elementer af designet, og ikke nødvendigvis samtlige små detaljer.



Figur 4 - Et typisk sequence diagram

## Valg af UML værktøj

Enhver håndværker ved, at man ikke nødvendigvis bliver en god håndværker ved at have det rigtige værktøj. En dårlig håndværker vil stadig lave dårligt håndværk, selv om han har det bedste værktøj penge kan købe. Men godt værktøj kan på den anden side gøre en god håndværker mere produktiv. Valget af det rigtige UML værktøj er da derfor også vigtigt. Ikke så vigtigt som at sikre at den håndværkmæssige kunnen er på plads i udviklingsteamet, men vigtigt nok til at bruge lidt tid og resurser på at undersøge alternativer.

Der findes i dag et hav af forskellige UML værktøjer på markedet. Licenspriserne varierer lige fra få tusinde kroner til ca. 100 gange så meget. Den høje ende af prisspektret domineres ikke overraskende af markedslederne IBM Rational, I-Logix Rhapsody og Borland Together, men der findes en lang række mindre firmaer, som udbyder langt billigere værktøjer af overraskende god kvalitet, og det er min anbefaling, at man i hvert fald i den første fase går efter de billigere modeller.

Man bør ikke vælge et værktøj, før man har testet mindst tre-fire stykker. Mange værktøjer findes i en såkaldt 'Community Edition', som er en skrabet udgave, der typisk kun må anvendes til ikke-kommercielle projekter. Men de kan være udmærkede til at få en fornemmelse af hvordan værktøjet er at arbejde med. Andre værktøjer kan fås i en fuldt funktionel men tidsbegrænset demo-udgave.

På min hjemmeside (<http://www.meconsult.dk>) kan man finde en diskussion af vigtige parametre ved valget af UML værktøj. Her kan man også finde en oversigt over UML værktøjer.

## Succesfaktorer

I den type virksomheder som denne artikel retter sig imod, kommer ønsket om at anvende UML typisk 'nede fra' i organisationen, og er sjældent drevet af et strategisk initiativ fra ledelsen. Netop derfor er det vigtigt at opnå et fast tilsagn fra ledelsen med hensyn til tidsforbrug og budget, inden man starter på UML introduktionen.

Al begyndelse er svær, og koster både tid og penge. Og hvis ledelsen kun har givet et valent tilsagn, som glemmes i samme sekund projektet er bare en smule presset rent tidsmæssigt, er UML initiativerne dømt til fiasko på forhånd.

De færreste mennesker kan lære nye komplekse ting helt på egen hånd, uden at modtage en eller anden form for struktureret vejledning. Uddannelse af projektdeltagerne er derfor også en vital ting. Såfremt projektet ikke råder over en person med tidligere UML erfaring, bør det overvejes at tilknytte en ekstern UML coach i en periode.

Det er desuden tilrådeligt at starte med UML aktiviteterne i et pilotprojekt, for at træne både ledelse og medarbejdere i de nye teknikker. Modsat hvad mange tror, behøver pilotprojektet ikke bestå af 100% nyudvikling. Man kan sagtens benytte et vedligeholdelses- eller featureudvidelsesprojekt som UML pilotprojekt, og så nøjes med at benytte UML til design af de nye faciliteter. Ofte afvises denne mulighed med argumenter om, at 'det ikke kan betale sig, da projektet jo allerede er designet, og nu bare skal udvides'.

Men formålet med et UML pilotprojekt er ikke at opnå en isoleret gevinst på dette projekt, men at give medarbejderne en træning i UML modelleringsteknikker. Gevinsten vil typisk opnås på de efterfølgende projekter. Faktisk kan det være en fordel at indføre UML på denne måde, da det mindsker antallet af ubekendte i projektet i forhold til et nyudviklingsprojekt, og dermed tillader deltagerne at fokusere mere på UML delen.

Det er meget tilrådeligt på forhånd at definere et sæt af klare succeskriterier for pilotprojektet, såsom hvilken type dokumentation man ønsker, med hvilken detaljeringsgrad, tilfredshedsgraden med de indkøbte værktøjer, m.m.



Figur 5 - Success-faktorer for anvendelse af UML

## Typiske faldgruber

Der er desværre adskillige måder hvorpå éns første UML projekt kan fejle. I det følgende har jeg prøvet at opstille de værste faldgruber, som jeg oplever dem:

**Kulturelle barrierer:** Den embeddede verden er traditionelt bundet til en ikke-objektorienteret design- og implementationsmetodik. Anvendelse af objektorienterede metoder og sprog er ikke helt så udbredt som i andre brancher, og betragtes mange steder med mistro og fordomme. For at imødegå dette problem er det vigtigt at der fra starten sættes ind med både uddannelse og holdningsbearbejdelse, samt at man sørger for at UML pilotprojektet bemannes med de rigtige folk, som ikke er bange for at lære noget nyt.

**For store ambitioner:** Det er nemt at knække halsen på at ville det hele fra starten; benytte alle diagramtyper, lave komplet modellering med kodegenerering, m.m., samtidigt med at man skal lære at bruge selve modelleringssproget og et nyt værktøj. Det kan næsten kun gå galt. Den bedste måde at imødegå dette på er, som ovenfor beskrevet, at starte med at bruge UML som skitseværktøj som led i en gradvis introduktion.

**At gøre tingene rigtigt i stedet for at gøre de rigtige ting:** Udviklings-teams som benytter UML for første gang kan nemt falde for fristelsen til at bruge megen tid på at diskutere hvad 'korrekt UML' er. Dette kan jeg ikke advare nok imod, da det er et eklatant spild af tid og ressourcer. Groft sagt er alt hvad der hjælper projektet til at nå de opstillede mål pr. definition 'korrekt UML'. Selvfølgelig bør man af hensyn til senere vedligeholdelse ikke opfinde en masse proprietære udvidelser, men fokus skal altid være på designet og opgaven, ikke på den 100 % syntaktisk korrekte anvendelse af UML.

**Licenskarrighed:** Mindre danske virksomheder har traditionelt aldrig været særligt motiverede for at bruge penge på værktøjer til softwareudvikling. Og da specielt high-end UML værktøjer

kan være ganske kostbare, kan det være fristende at købe lidt færre licenser end man egentlig synes man har brug for. Det er imidlertid at lægge gift ud for projektet fra starten. Hvis anvendelsen af UML skal slå rod i organisationen, er det imperativt at alle projektdeltagere har adgang til værktøjerne når de har behov derfor. Hvis ledelsen ikke vil slippe penge til det dyre værktøj, så køb en billigere model, så alle kan få en licens.

**Design-ændringer dokumenteres ikke:** Hvis et design skal have værdi for fremtidige projekter og kolleger, er det vigtigt at det i rimelig grad afspejler det implementerede. Dette problem opstår typisk når dele af designet ændres efter at implementationsfasen har været i gang i et stykke tid, og projektet måske er i tidsmæssige problemer. Her er det nemt at 'komme til' at glemme at føre designændringer tilbage til dokumentationen. For at imødegå dette problem, skal man for det første sørge for at den ovenfor nævnte licensproblematik ikke kan bruges som undskyldning, og dernæst skal den ligeledes ovenfor nævnte holdningsbearbejdelse sørge for at folk virkelig føler at design-feedback er vigtigt.

## Valg af udviklingsmetode

UML alene gør det ikke. UML er en notationsstandard til brug for objektorienteret analyse og design, men giver i sig selv ikke nogen anvisninger på hvordan man i øvrigt kommer helskindet gennem et udviklingsforløb. Anvendelsen af UML skal derfor integreres med virksomhedens overordnede udviklingsproces og gerne anvendes i rammen af en udviklingsmetode.

Det er en udbredt misforståelse at anvendelsen af UML kun kan ske gennem en moderne iterativ metode, såsom Rational Unified Process (RUP). Der er dog intet der forhindrer én i at benytte UML i relation til virksomhedens eksisterende metode, som ofte vil være mere eller mindre baseret på den traditionelle vandfaldsmodel. At der så er ganske gode grunde til at se endog meget kritisk på bemeldte vandfaldsmodel er en helt anden sag, men det bør man ikke blande sammen med sin UML introduktion, igen for at undgå at lave for mange tiltag på en gang.

## Brug det rigtigt

UML kan anvendes til design af softwaresystemer af enhver art, men bør ikke anvendes ukritisk. Brugt rigtigt, kan anvendelse af UML hjælpe med at designe bedre softwaresystemer. Brugt forkert, kan det rent faktisk resultere i dårligere design end hvis man havde designet på traditionel vis.

For en projektleder, der ønsker at indføre anvendelse af UML i et konkret projekt, er udfordringen at tilpasse ambitionsniveauet til virksomhedens modenhedsniveau, medarbejdernes faglige og holdningsmæssige baggrund og det konkrete projekts tidsmæssige rammer.

*Denne artikel blev bragt i nyhedsmagasinet Elektronik & Data nr. 5, april 2006.*