

# Baggrundsmateriale til novellen ”Skattens sorte æske”

Det følgende er udarbejdet til brug sammen med novellesamlingen ”Den chipløse mand og andre noveller om sex, magt og informationsteknologi” af Georg Strøm

Julia Lawall og Georg Strøm

31. juli 2008

Problematikken i novellen er desværre ikke helt urealistisk.

Ganske vist troede jeg ikke at der var fejl i softwaren til selvangivelserne, før jeg i år ville indtaste min selvangivelse i Skats nye web interface. Der stod i forvejen en hel række tal som allerede var indberettet fra forskellige steder. Tallene så ud til at stemme med mine egne papirer, så jeg godkendte dem. Derefter fik jeg besked om at tallene ikke stemte med dem som Skat i forvejen havde registreret. Jeg fik altså at vide at deres egne tal var forkerte - og jeg blev bedt om at give en forklaring på forskellen.

Computerbladet Version 2 beskriver i en artikel den 30. november 2007 hvordan man stadig anvender programmer som er skrevet for 30 år siden i Cobol eller PL/1 – to programmeringssprog som ikke anvendes mere. I følge artiklen kan man godt ændre i programmerne, men der er ikke længere nogen som kan overskue konsekvenserne når man begynder at pille i de gamle programmer.

I novellen optræder udtrykket ”legacy”. Det er software som er overtaget fra en tidligere teknologi eller et tidligere system, og som man så skal have til at fungere i det nye.

## Principper for test af software

Når man udvikler et program, er det vigtigt at man sikrer sig at det faktisk opfører sig som det skal. For at gøre det, behøver man en specifikation af hvordan det er meningen programmet skal opføre sig. Desuden behøver man en metode, så man kan undersøge om programmet opfører sig som det skal.

Når man fodrer nogle bestemte oplysninger ind i programmet – giver det nogle bestemte input – skal der komme de rigtige oplysninger ud af det – de såkaldte output.

En almindelig metode er test af programmet. Den er baseret på at man studerer specifikationen, og finder alle de tilfælde hvor input skal behandles forskelligt. I et program til beregning af skat, vil der for eksempel være forskellige tilfælde som afhænger af personens indtægt. Hvis den er lille, kan han eller hun for eksempel få et frikort.

Når man har identificeret alle de forskellige tilfælde, laver man nogle input som dækker alle de forskellige tilfælde. Det kan for eksempel være en person som kun tjener ti tusind kroner om året, så han eller hun skal have frikort. Derefter fodrer man de forskellige input ind i programmet, og ser om de alle sammen giver de output som de skal.

Når man har testet programmet, skal man også finde og rette fejlene. Det kan være svært i et stort program. Derfor vil man ofte gennemføre en såkaldt modultest før man tester hele programmet. Her tester man at hver modul eller del af programmet giver de ønskede output, når man fodrer det med nogle bestemte input. Når man så har sikret at alle moduler fungerer korrekt, kan man sætte dem sammen, og derefter teste at det samlede program giver det rigtige output.

Det lyder meget enkelt, men i praksis er det vanskeligt. Det er svært at bestemme alle de mulige tilstande som kan opstå når de forskellige input vekselvirker med hinanden. Der kan for eksempel være tale om at en person er gift, så et ubenyttet fradrag kan overføres til ægtefællen, der kan være transportfradrag, regler for rentefradrag, foruden en forskellig beregning af skat i forhold til indtægten. Derfor er det meget svært at komme op med test input som dækker alle de forskellige kombinationer af input som er mulige.

I værste fald kan der være en bestemt kombination af input som man har overset, da man skrev programmet, så det ikke kan behandle dem, og hvis de optræder, kan det betyde at programmet simpelthen stopper. Et andet problem, er at man er nødt til at antage at der ikke er uregelmæssigheder som går ud over de tilfælde som kan beskrives ud fra specifikationen. Hvis et skatteprogram af en eller anden underlig grund udsteder frikort til alle som tjener præcis en million kroner om året, vil man næppe have tænkt på at lave et input som tester det.

For at løse de problemer har man udviklet model-testning. Her bruger man specifikationen sammen med en model af alle de mulige måder som programmet kan køre på. Den grundlæggende ide er så at man viser at alle mulige veje gennem programmet tilfredsstillende specifikationen. Hvis man for eksempel tager det tilfælde hvor en person får frikort, så er der samtidig en række regler, for eksempel om transportfradrag og topskat, som ikke er aktuelle. Her vil modeltesten sikre at man ser bort fra dem, og for eksempel ikke giver transportfradrag til en person som slet ikke betaler skat. Her vil man altså checke alle de tilfælde som er mulige i programmet, og ikke kun dem som er med i specifikationen.

Model testning lyder som den ideelle løsning når man skal teste om et program faktisk svarer til specifikationen. Imidlertid er det alt for tidskrævende at lave en modeltest som går gennem alle de måder som programmet kan køre på. Sådant en test vil kræve at modellen gengiver hele koden til programmet, og gennem testningen må man gå gennem hver eneste mulige vej gennem koden. Det betyder at der hurtigt bliver så mange mulige kombinationer af input og så mange forskellige tilstande at holde styr på, at det er alt for tidskrævende og kræve så meget hukommelse, at det ikke kan køre på en almindelig computer. Derfor vil man ofte vælge kun at teste en abstraktion af programmet – man vil altså ignorere alle de dele som man mener er uvæsentlige. I eksemplet med et skatteprogram, kan man vælge kun at ville teste selve beregningen af skattens størrelse, og udelade alle de funktioner som bruges til at hente forskellige tal, udskrive blanketter med mere. Problemet er bare, at man skal vælge den

rigtige abstraktion, og det kan være lige så svært som at vælge de rigtige input til en test. Når man laver en abstrakt model af skatteprogrammet, kan det for eksempel vise sig at man er kommet til at udelade de funktioner som er nødvendige hvis en person har fået ændret skattekortet. Desuden vil modeltesten så ikke sige noget om det originale skatteprogram, men kun om den abstrakte model. Derfor er man nødt til at vise at den abstrakte model dækker alle de vigtige dele af selve programmet.

Model test blev først udviklet til test af integrerede kredse, og er først for nylig blevet anvendt til test af programmer. Oprindeligt var modellerne meget abstrakte og blev lavet med håndkraft. For nylig er der så udviklet programmer hos Microsoft, Berkeley og andre steder som automatisk kan lave en model af et program som er skrevet i et programmeringssprog som C eller Java. Disse programmer betyder at man undgår menneskelige fejl i forbindelsen mellem programmet og modellen, men det er stadig begrænset hvor komplicerede programmer de kan lave modeller af, og hvilke typer af problemer de kan finde.