Building Product Populations with Software Components

Ommering, Robbert Christiaan van

Samenvatting

In veel consumentenproducten zijn tegenwoordig ‘kleine’ computers ingebouwd voor de interne besturing van het product. Net als in de wereld van de persoonlijke en zakelijke computer bevatten nieuwere producten krachtigere computers dan hun voorgangers. De capaciteit van de ingebouwde computers verdubbelt elke twee jaar, en volgt daarmee nauwgezet de Wet van Moore.

De televisie was een van de eerste consumentenproducten met een ingebouwde computer. In 1978 had een topmodel Philips TV een microprocessor met 1 kilobyte programmageheugen. Ruim tien jaar later telde een televisie 64 kilobyte aan geheugen, en na weer tien jaar meer dan een megabyte. Andere voorbeelden van consumentenproducten met ingebouwde computers zijn video recorders, DVD spelers en recorders, telefoons (vast en mobiel), auto’s, fotocamera’s, magnetrons, koelkasten, wasmachines, stofzuigers, scheerapparaten, koffiezetmachines, en de lijst is groeiende.

Voor bedrijven zoals Philips levert dit de volgende uitdagingen:

- De complexiteit van de programmatuur in een individueel product stijgt, en het wordt steeds moeilijker om de gewenste hoge kwaliteit te handhaven.
- Een bedrijf verkoopt niet één product, maar een familie van sterk verwante producten. Hierbij is het gewenst de programmatuur zo te ontwerpen dat zo veel mogelijk gedeeld kan worden tussen leden van de familie.
- De markt wordt steeds dynamischer, daarom is het belangrijk om de tijd van het ontwerpen en maken van een product steeds kleiner te maken.
- Tenslotte leveren bedrijven als Philips niet slechts één familie van producten (televisies), maar een aantal families (CD/DVD spelers, geluidsversterkers) die onderling weer gerelateerd zijn. Delen van programmatuur tussen deze families leidt tot een reductie van ontwikkelkosten, maar maakt ook combinatieproducten mogelijk, zoals een TV met ingebouwde DVD speler.

Dit proefschrift bouwt op drie stromingen in de wetenschap en technologie: een verhoogde aandacht voor de *architectuur* van computerprogramma’s, het ontstaan van technieken om *software componenten* te maken, en het systematisch opzetten van *productielijnen* voor het maken van programmatuur. Het probeert daarbij de volgende vragen te beantwoorden:

- Kunnen we de architectuur van computerprogramma’s expliciet maken, met garantie van consistentie tussen architectuur en implementatie, opdat we in architectuurtermen over de implementatie kunnen redeneren?
- Kunnen we families en ‘populaties’ van producten maken met behulp van software componenten (lees een ‘populatie’ als ‘een familie van families’)?

- Kunnen we profiteren van moderne technieken om software componenten te maken zonder dat onze systemen daardoor duurder of trager worden?
- Wat voor een consequenties heeft dit op het te volgen ontwikkelproces en de daarbij benodigde ontwikkelorganisatie?

Eerst beschrijft dit proefschrift een wiskundige techniek om software architecturen te modelleren. Hierbij is een automatische controle op interne consistentie en op consistentie met de implementatie mogelijk met behulp van een kleine verzameling gereedschappen. Deze techniek is door mij en diverse collega's ontwikkeld en succesvol toegepast op een verscheidenheid aan systemen. Intrinsiek nadeel van de methode is dat inconsistentie altijd achteraf gevonden wordt, en dan is het vaak te laat in het ontwikkelproces om de inconsistentie op te lossen.

Daarom introduceren we een taal waarin de architectuur beschreven kan worden en van waaruit delen van de implementatie gegenereerd kunnen worden zodat er per definitie consistentie is. Tevens dient deze taal als een component technologie waarmee het mogelijk is om vanuit dezelfde componenten verschillende producten te bouwen. De taal is zo ontworpen dat de producten niet duurder of trager gemaakt worden. We noemen nog enige andere contributies van ons werk:

- De taal ondersteunt *onafhankelijke ontwikkeling*: een component behoeft niet langer in een specifieke context ontwikkeld te worden, maar kan relatief zelfstandig evolueren. Dit vergt technieken om veranderingen zo te implementeren dat nieuwere versies van de component samen met oudere versies van andere componenten gebruikt kunnen worden.
- De taal benadrukt *compositie*, terwijl onderzoekers aan productielijnen vaak *variatie* benadrukken. Compositie kan gezien worden als een ruimere vorm van variatie, en wordt noodzakelijker naarmate het bereik van de familie(s) groter wordt.
- De klassieke techniek om meerdere producten te bouwen is *configuratie beheer*. Wij menen dat het verstandig is om variatiebeheer in de architectuur op te nemen in plaats van het apart op te lossen.

Vervolgens beschrijven we hoe we de taal en bijbehorende methodiek binnen Philips toepassen op de ontwikkeling van software voor middenklasse en topmodel televisies. Dit vergt een aanpassing van het software ontwikkelproces en van de ontwikkelorganisatie, die van oudsher geoptimaliseerd zijn voor het maken van één product. Hierbij zijn we wel geholpen doordat de elektronica, mechanica en optica afdelingen al langer in families denken.

We bespreken één (uitgebreid) voorbeeld van een ontwerp dat compositie mogelijk maakt in een deel van de programmatuur dat altijd als ingewikkeld en slecht componeerbaar werd beschouwd. Daarmee is het laatste woord over compositie nog lang niet gezegd, en wij nodigen de lezers dan ook uit om het werk dat in dit proefschrift beschreven is, toe te passen, te verbeteren en voort te zetten.