procedureel programmeren > objectgeoriënteerd programmeren

"OOP is een hype, gaat in tegen het KISS-principe, en ik vind het een onplezierige werkwijze."

Inleiding

In dit document leg ik uit waarom ik procedureel programmeren (PP) verkies boven objectgeoriënteerd programmeren (OOP). Andere programmeerparadigma's (programmeermethodologieën) als functioneel of logisch programmeren laat ik buiten beschouwing. Alle paradigma's en talen hebben voor- en nadelen, en *"ieder nadeel heb zijn voordeel"*, maar ik zet enkel uiteen waarom ik er principieel voor kies om niet-objectgeoriënteerd te werken. Ik ga niet in op de dooddoener *'het geschikte paradigma moet worden gekozen bij het soort probleem'* aangezien ik kies voor de weg van de specialisatie.

Met dit document wil ik geen discussie op gang brengen. Het is ook niet mijn bedoeling anderen te overtuigen van de juistheid van mijn keuze, noch heb ik het gevoel die keuze te moeten verdedigen. Iedereen kan voor zichzelf een (of meer) paradigma('s) kiezen op basis van onder meer eigen ervaringen en allerlei voor en tegens die op diverse websites uitgebreid worden bediscussieerd. Af en toe wordt mij gevraagd waarom ik voor PP kies en het is makkelijk als ik dan een link kan geven naar dit document.[1]

Programmeerervaring

Programmeren is al zo'n vijfentwintig jaar een grote hobby van mij. Ik begon met BASIC, op een Philips P2000T homecomputer. Vervolgens gebruikte ik Pascal (en assembly) onder DOS, op - als eerste - een 286. Ik bracht destijds een handjevol programma's uit in publiek domein. Daarna begon ik C te gebruiken onder Windows 95. Aangezien in C geschreven *exploit*-code veelal bestemd was voor compilatie onder GNU/Linux maakte ik in 1997 de overstap naar dat besturingssysteem. Ik hield me destijds namelijk bezig met hacken en *phreaken* (ging onder meer naar Hacking In Progress) en had net kennisgemaakt met GNU/Linux. Tijdens het eerste jaar van mijn studie Hogere Informatica zag ik de *'coole'* studenten dat gebruiken. Tot op de dag van vandaag gebruik ik C (en GNU/Linux), veelal in combinatie met library's als GTK+/GNOME, Xlib en SDL. Sinds mijn overstap naar GNU/Linux breng ik programma's uit onder de GPL. Daarnaast heb ik over de jaren de nodige websites en webapplicaties geschreven in PHP/JavaScript/SQL, soms gecombineerd met JavaScript-library's als jQuery. Af en toe gebruik ik scripttalen als Expect en shellscript. Voor mijn afstudeerproject gebruikte ik Borland Delphi.

Met OOP maakte ik kennis tijdens mijn studie, waardoor ik kort met Java en C++ werkte. Voor werkgevers heb ik weinig met OOP gedaan. Ik moest me verdiepen in enige Magento(-gerelateerde) code; Zend Framework. Privé heb ik de nodige objectgeoriënteerde code van anderen doorgespit, bijvoorbeeld als ik die (deels) wilde gebruiken/wijzigen of als er compilatieproblemen optraden.

Niet-significante verschillen

Er is een aantal punten waarop OOP en PP niet-significant verschillen. Zo werken voor grote, complexe projecten met gedegen bouwplannen zowel OOP als PP (denk aan bijvoorbeeld de Linux kernel, Apache, Samba en Drupal). Ook is de herbruikbaarheid van code niet gebonden aan een paradigma: die hangt af van hoe men programmeert. Dat OOP resulteert in meer geheugen-/CPU-gebruik/overhead en (daardoor) langzamer is: tegenwoordig is dat niet zo'n issue meer. Ook is er geen significant verschil in productiviteit tussen OOP en PP benaderingen. Of OOP uitgaat van te simplistische modellen van de wereld, of de wereld uit objecten bestaat, of de wereld opgedeeld kan worden in classes en subclasses: voor mij zijn die zaken irrelevant voor de keuze tussen OOP en PP.

---

1 http://www.norbertdejonge.nl/pdf/PPvsOOP.pdf

PP > OOP

"OOP is een hype, gaat in tegen het KISS-principe, en ik vind het een onplezierige werkwijze."

OOP is een hype...

Nieuwer is niet per se beter. In mijn beleving wordt OOP door te veel managers en studenten, en zelfs professionals, gezien als programmeren 2.0. OOP werd een dominant paradigma ná PP, in begin/medio jaren 1990. OOP wordt (daardoor) gezien als een extra set regels en voorschriften bovenop PP. Een uitgebreide soep van zoemwoorden en vage concepten, zoals *"responsibility-driven design"*, *"encapsulation"*, *"decoupling"*, enzovoort, die overigens voor iedereen wat anders betekenen, draagt daartoe bij. In werkelijkheid is OOP geen programmeren 2.0. Als het hoogst haalbare de top van de Mount Everest is klimt de programmeur met OOP niet hoger op die berg maar neemt de programmeur slechts één van de vele paden (paradigma's). OOP als pad krijgt overmatige aandacht. Managers laten zich mogelijk door die hype leiden en kiezen voor OOP op basis van simpele (cliché) tekstboekvoorbeelden. Te veel studenten zien OOP onterecht als een schepje bovenop PP door de manier waarop opleidingen en coachingstrajecten (*"kennis up-to-date maken"*) focussen op het OOP paradigma en minder aandacht besteden aan alternatieven.

...gaat in tegen het KISS-principe...

OOP gaat in tegen het KISS-principe (*"Keep it simple, stupid"*). Een programmeur met voldoende vaardigheden en kennis van de juiste technieken bereikt met PP hetzelfde resultaat als met OOP. Het is daarom níét logisch om te kiezen voor het complexere paradigma dat meer valkuilen heeft en waarbij er een grotere kans is op doolhof-code. Het is wél logisch om te kiezen voor het paradigma dat simpel te leren is, simpel te gebruiken is, en code oplevert die simpel te lezen is. Het is onlogisch om voor de moeilijke oplossing te kiezen wanneer die vermeden kan worden en onnodig is. Bij PP is het verloop en de structuur van programma's makkelijker te volgen. Functies/procedures zijn makkelijk (in isolatie) te debuggen; eventuele problemen zijn snel gevonden. Het modulaire karakter maakt het geheel flexibel. Als code van anderen gebruikt wordt, zoals library's, is die makkelijk door te nemen en te wijzigen.

...en ik vind het een onplezierige werkwijze.

Ik vind OOP onplezierig. Programmeren is een hobby van mij en ik vind het leuk om te doen. OOP ontneemt mij dat plezier. Waarom? Onder meer omdat het ingaat tegen het KISS-principe.

Daarom kies ik er principieel voor om niet-objectgeoriënteerd te werken.