De programmeur als kunstenaar

Bart Barnard

juni 2015

Op het gebied van de design en kunst is een toenemende samenwerking met computerprogrammeurs waar te nemen. Hoe goed deze samenwerking over het algemeen ook uitpakt en hoe bijzonder of mooi de resultaten hiervan ook zijn, het blijft toch nog vaak hangen in de gedachte van de kunstenaar als creatieveling en de programmeur als uitvoerder. In dit artikel en de bijhorende presentatie breek ik een lans voor de gedachte dat het dagelijks werk van deze laatste, het daadwerkelijk *schrijven van programmacode* zich beter laat vergelijken met het werk van een (autonoom) kunstenaar dan met dat van een ambachtsman.

Zeker sinds de Renaissance hoeft de autonoom kunstenaar zich niets gelegen laten liggen aan vorm of functie van zijn producties. Hij kan in zijn werk al zijn creativiteit en individualiteit kwijt; het resultaat hiervan draagt ondubbelzinnig zijn handtekening. Hetzelfde geldt voor de muziek en voor de literatuur: Mahler is anders dan Mozart, Marsman anders dan Multatuli. Maar dit geldt ook voor de programmeur. De plasticiteit en complexiteit van het domein maakt het mogelijk dat hij in de uiteindelijke implementatie zijn eigen stijl kwijt kan. Net als kunstkenners kunnen veel programmeurs zonder moeite aan de stijl van de programmacode identificeren wie van zijn collega's dat specifieke stuk heeft geschreven.

Net als het autonome kunstwerk wordt ook van programmacode en programmeertalen gezegd dat ze *mooi* zijn. Programmeurs hebben vaak een duidelijke voorkeur voor een programmeertaal of een *framework* en proberen

ze ook hun collega's te overtuigen van hun gelijk. Argumenten om de ene boven de andere taal te verkiezen beroepen zich vaak op de *schoonheid* van deze taal. Bepaalde implementatie-details worden vaak met een beroep op *esthetische kwaliteiten* gelegitimeerd; vrijwel zonder uitzondering vinden programmeurs bijvoorbeeld recursieve implementaties beter, want mooier, dan lineaire. Net als kunstenaars worden goede programmeurs geroemd om de *schoonheid* van hun werk.

Van veel kunstwerken is maar een heel klein onderdeel zichtbaar. Dit kleine onderdeel vormt als het ware een soort *herinnering* aan het complexe achterliggende systeem; iets hoeft niet per se waarneembaar te zijn om als kunstwerk gekwalificeerd te kunnen worden. Ook dit geldt voor programmacode: datgene dat uiteindelijk van het werk zichtbaar is, is maar een minieme verschijningsvorm van het achterliggende complex: de eindgebruiker ziet de app of de site en is zich maar ten dele bewust van dat wat er achter ligt. Wanneer de app of de site helemaal klaar en operationeel is, is de noodzaak verdwenen ooit nog naar de achterliggende programmacode te kijken.

Het kunstwerk ontstaat in de dialoog tussen de kunstenaar en de materie. Bekend is de uitspraak van Michelangelo dat het volstaat om het overbodige marmer weg te slaan om het beeld tevoorschijn te laten komen. Een kunstenaar weet pas hoe een werk er uit komt te zien wanneer hij ermee aan de gang gaat. Dit vergelijkt zich heel goed met de manier waarop een programmeur werkt. Net als de kunstenaar komt de programmeur onvoorziene en onvoorzienbare zaken op zijn pad tegen die hem dwingen het werk anders in te richten of anders op te zetten. Net als de kunstenaar gaat de programmeur een dialoog aan met zijn werk; de weerbarstigheid van de materie (verf of marmer in het geval van de kunstenaar, hard- en software in het geval van de programmeur) heeft tot gevolg dat het nooit op voorhand volledig duidelijk kan zijn wat er exact gaat gebeuren wanneer de eerste regels code worden geschreven of nieuwe functionaliteit aan een legacy-systeem wordt toegevoegd. De programmeur gaat op in zijn werk en vormt hiermee een creatieve eenheid.

Als docenten en begeleiders moeten we onze studenten derhalve voorbereiden op *de verandering*, op het feit dat software-ontwikkeltrajecten zich net zo min volledig laten plannen als het maken van een kunstwerk. We moeten kijken naar de creatieve opleidingen en industrieën, onderzoeken welke aspecten daarvan we in onze curricula kunnen overnemen, en inventariseren op welke punten hiermee samenwerking mogelijk is. Want de computer-expert die zichzelf louter profileert als technologisch ingenieur, en het hele creatieve moment dat inherent in dit werk aanwezig is uit het oog verliest, ontneemt zichzelf en zijn omgeving de mogelijkheid werkelijk grootste daden te verrichten.