



Stichting NIOC

Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2025, gehouden op donderdag 27 maart 2025 jl. en georganiseerd door Hogeschool Windesheim). Bij elkaar zo'n 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op 18 maart 2027 in Arnhem en wordt georganiseerd door HAN University of Applied Sciences.

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

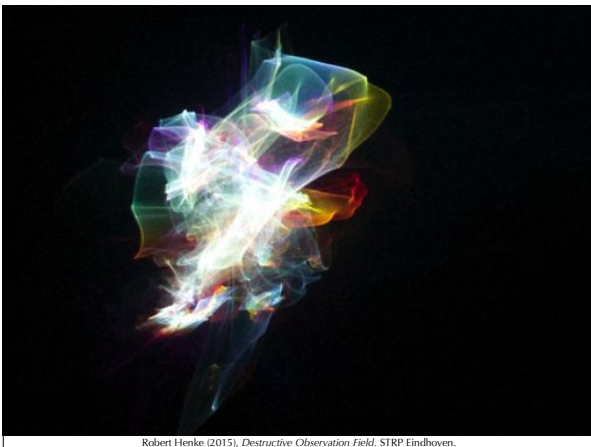
Bart Barnard
*De programmeur
als kunstenaar*

二零十五年四月二十四日



Studio Roosegaarde (2011), *Dune 4.2*, CBK Rotterdam.

Niet hebben over de programmeur *in dienst van* de kunstenaar, maar over de programmeur *als* kunstenaar: zijn dagelijkse werk vergelijken met dat van een kunstenaar.



Robert Henke (2015), *Destructive Observation Field*, STRP Eindhoven.

Ook niet over de vraag wat kunst is: appelleer aan de werk- en productiewijze van prototypische kunstwerken.
Aan de hand van vier aspecten: individualiteit, schoonheid, (on)zichtbaarheid en werkwijze.

persoonlijkheid



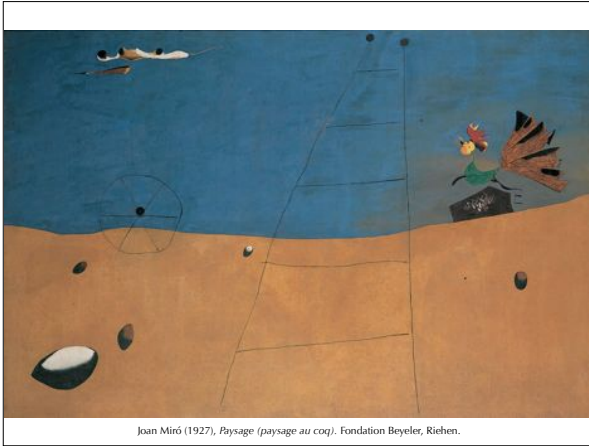
Kom met geometrisch motief, ca. 1550-1400 v.Chr. Rijksmuseum van Oudheden, Leiden.

De ambachtsman heeft maar weinig ruimte om zijn persoonlijke stempel op zijn werk te drukken.



Giotto (1320 circa). Assunzione di san Giovanni Evangelista. Cappella Peruzzi.

Zeker sinds de Renaissance komt de persoonlijkheid steeds meer terug in het werk.



Joan Miró (1927), *Paysage (paysage au coq)*. Fondation Beyeler, Riehen.

This family knew that Dresden was gone. Those with eyes had seen it burn and burn, understood that they were on the edge of a desert now. Still – they had opened for business, had polished the glasses and wound the clocks and stirred the fires and waited and waited to see who would come.

Kurt Vonnegut, *slaughterhouse 5*. p.149

Bypasses are devices which allow some people to dash from point A to point B very fast whilst other people dash from point B to point A very fast. People living at point C, being a point directly in between, are often given to wonder what's so great about point A that so many people from point B are so keen to get there...

Douglas Adams, *The Hitch Hiker's Guide to the Galaxy*. p.19

Dat geldt natuurlijk niet alleen voor schilderijen. Ook in literatuur is persoonlijk stijl belangrijk: je kunt de auteurs herkennen aan hun 'handschrift'

@author

We are authors

The `@author` field of a Javadoc tells us who we are. We are authors. And one thing about authors is that they have readers. Indeed, authors are responsible for communicating well with their readers. The next time you write a line of code, remember you are an author, writing for readers who will judge your effort.

Martin 2009, pp.13f.

Ook programmeurs zijn auteurs.

```
$ find . -name *Helper.java | xargs grep -l "extends .*Helper"

$ find . -name *Helper.java -exec grep -l "extends .*Helper" {} \;
```

```
private EntityManager getEntityManager(String rekeningnummer) {
    int prefix = Integer.valueOf(rekeningnummer.substring(0, 1));
    switch (prefix) {
        case 1:
            return em1;
        case 2:
            return em2;
        default:
            return null;
    }
}
```

Code: Johan Mulder

```
private EntityManager getEntityManager(String rekNr)
{
    int check = Integer.valueOf(rekeningnummer.charAt(0));
    return (check == 1) ? em1 : em2;
}
```

Code: Vincent Vogelesang

The diagram illustrates the Design Patterns, categorized into three main groups: Behavioral, Structural, and Creational. A central box labeled "Design Patterns" is connected to these three categories.

- Behavioral** (labeled "CSYS"):
 - Visitor**: Includes "Visitor" and "Concrete Visitor".
 - Observer**: Includes "Observer" and "Concrete Observer".
 - Command**: Includes "Command", "Invoker", "Concrete Command", and "Concrete Invoker".
 - Strategy**: Includes "Strategy", "Context", "Concrete Strategy", and "Concrete Context".
 - Iterator**: Includes "Iterator" and "Concrete Iterator".
 - Chain of Responsibility**: Includes "Chain of Responsibility" and "Concrete Chain of Responsibility".
 - State**: Includes "State" and "Concrete State".
 - Facade**: Includes "Facade" and "Concrete Facade".
 - Adapter**: Includes "Adapter" and "Concrete Adapter".
 - Decorator**: Includes "Decorator" and "Concrete Decorator".
 - Builder**: Includes "Builder" and "Concrete Builder".
 - Factory Method**: Includes "Factory Method" and "Concrete Factory Method".
 - Abstract Factory**: Includes "Abstract Factory" and "Concrete Abstract Factory".
 - Singleton**: Includes "Singleton" and "Concrete Singleton".
 - Proxy**: Includes "Proxy" and "Concrete Proxy".
 - Bridge**: Includes "Bridge" and "Concrete Bridge".
 - Composite**: Includes "Composite" and "Concrete Composite".
 - Decorator**: Includes "Decorator" and "Concrete Decorator".
 - Builder**: Includes "Builder" and "Concrete Builder".
 - Factory Method**: Includes "Factory Method" and "Concrete Factory Method".
 - Abstract Factory**: Includes "Abstract Factory" and "Concrete Abstract Factory".
 - Singleton**: Includes "Singleton" and "Concrete Singleton".
 - Proxy**: Includes "Proxy" and "Concrete Proxy".
 - Bridge**: Includes "Bridge" and "Concrete Bridge".
 - Composite**: Includes "Composite" and "Concrete Composite".
- Structural** (labeled "ACF"):
 - Facade**: Includes "Facade" and "Concrete Facade".
 - Adapter**: Includes "Adapter" and "Concrete Adapter".
 - Decorator**: Includes "Decorator" and "Concrete Decorator".
 - Builder**: Includes "Builder" and "Concrete Builder".
 - Factory Method**: Includes "Factory Method" and "Concrete Factory Method".
 - Abstract Factory**: Includes "Abstract Factory" and "Concrete Abstract Factory".
 - Singleton**: Includes "Singleton" and "Concrete Singleton".
 - Proxy**: Includes "Proxy" and "Concrete Proxy".
 - Bridge**: Includes "Bridge" and "Concrete Bridge".
 - Composite**: Includes "Composite" and "Concrete Composite".
- Creational** (labeled "AS"):
 - Singleton**: Includes "Singleton" and "Concrete Singleton".
 - Proxy**: Includes "Proxy" and "Concrete Proxy".
 - Bridge**: Includes "Bridge" and "Concrete Bridge".
 - Composite**: Includes "Composite" and "Concrete Composite".

The diagram also includes a "Design Patterns" box at the top, which is connected to the "Behavioral" and "Structural" categories. The "Design Patterns" box is also connected to the "Creational" category.

schoonheid



Schoonheid kan voorkomen in de natuur



Quentin Tarantino (1994), *Pulp Fiction* (shooting Brett scene).

Iets wat je mooi vindt *waardeer* je en je vertoont de neiging om daar telkens weer naar toe terug te keren. Zoals een film meerdere keren zien, of een muziekstuk meerdere keren beluisteren.

“Like beautiful people, beautiful works spark the urgent need to approach, the same pressing feeling that they have more to offer, the same burning desire to understand what it is.

Nehamas 2007, p.73

Je hecht belang aan wat je mooi en lelijk vindt. Dit bepaalt voor een groot deel wie je bent en in wat voor omgeving (groep) je je prettig voelt. Je hebt weinig voeling met mensen die jouw mening niet delen.

$$e^{i\pi} + 1 = 0 \quad \sum_n \frac{1}{n^s} = \prod_p \frac{1}{1 - \frac{1}{p^s}}$$

$$t' = \frac{t}{\sqrt{1 - \frac{v^2}{c^2}}} \quad a = \sqrt{\sum_{i=1}^N p_i (x_i - \mu)^2}$$

cf. Juarez 2010

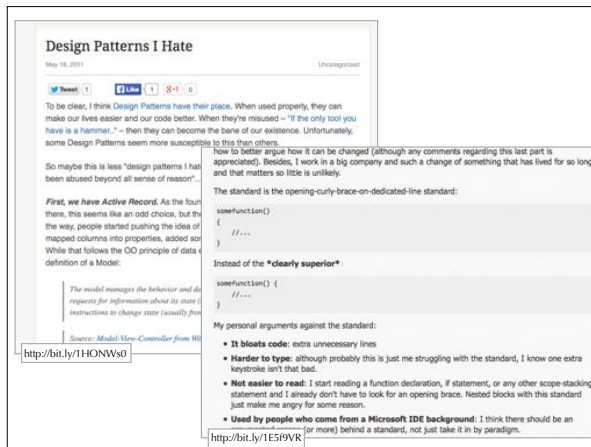
De schoonheid van een wiskundig of natuurkundig bewijs is belangrijk, maar uiteindelijk niet doorslaggevend (zie Juarez 2010, pp. 31ff). Je kunt niet door een appél op de schoonheid van een formule zijn waarheid / correctheid bewijzen.

Schoonheid zit in natuur, kunst (film) en wetenschap. De binding van de programmeur is echter groter dan de wiskundige, ondanks dat het in beide gevallen een esthetische euforie oplevert.

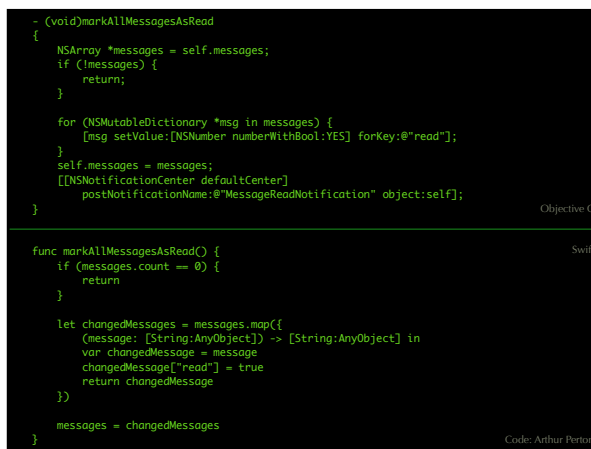
“From the programmer's point of view, the executive doesn't have any skin in the game, so obedience is problematic. The independent-minded software-engineer won't change his code because someone tells him to, regardless of the magnitude of their title.

Cooper 1999, p.118

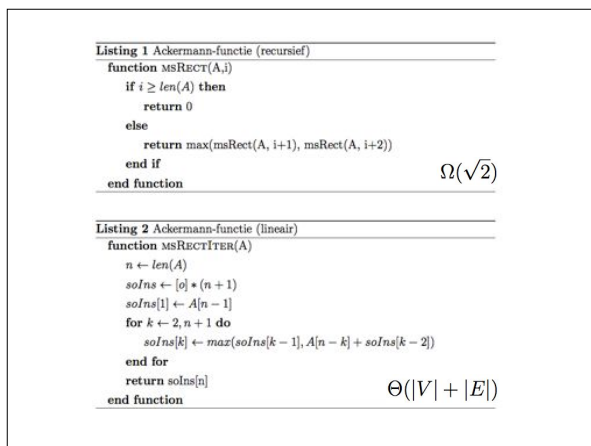
De autonomie van de programmeur.



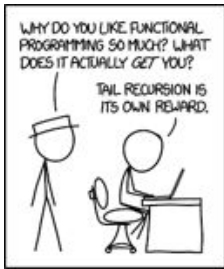
Programmeurs leggen dezelfde passie voor hun werk aan de dag als liefhebbers van een bepaald kunstwerk: mensen die het niet met hen eens zijn, zijn niet goed wijs.



Collega's gebruiken vaak esthetische argumenten om de ene taal of het ene framework te verkiezen boven het anderen. Bijvoorbeeld swift boven objective-c.



Dat geldt niet alleen voor programmacode. Hier een extreem voorbeeld, maar de schoonheid van Listing 1 is onafhankelijk voor zijn slechte performance.



Schoonheid zit in recursie. Veel collega's vinden recursieve altijd mooier / beter dan lineaire implementaties.

(on)zichtbaarheid

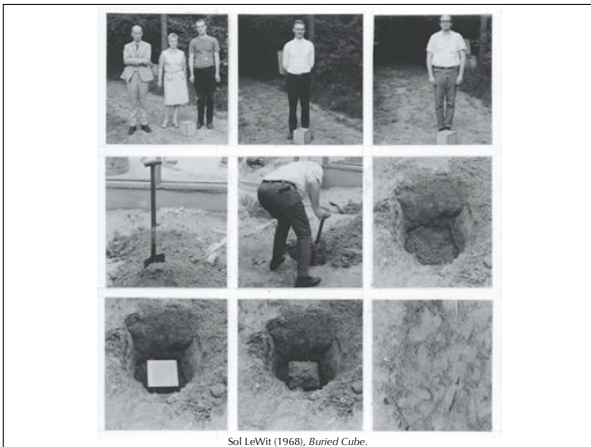


Kunst heeft vaak een zichtbare en een onzichtbare component. Je ziet de klodders verf zitten, maar vanuit een ander blikveld zie je gelijk en direct het schilderij.



Walter de Maria (1977), *Vertical Earth Kilometre*. Documenta VI, Kassel.

Dit geldt echter niet voor alle kunst. Vaak gaat het juist om de onzichtbaarheid.



Sol LeWit (1968), *Buried Cube*.

Work is visible *in principle*, but in practice, both the work and the box were never seen again.

Sol LeWit – Buried Cube (1968)



<https://www.youtube.com/watch?v=a5mVSLcksw>

Vaak gaat het niet eens om een ding, maar om een idee of een actie. Of een feit, zoals hier bij Wim Schippers.



Grove technieken zijn over het algemeen behoorlijk zichtbaar, zoals hier een stoomtrein. Met weinig moeite is ook de techniek (de stoomketel) achter het werk (het rijden) te begrijpen.

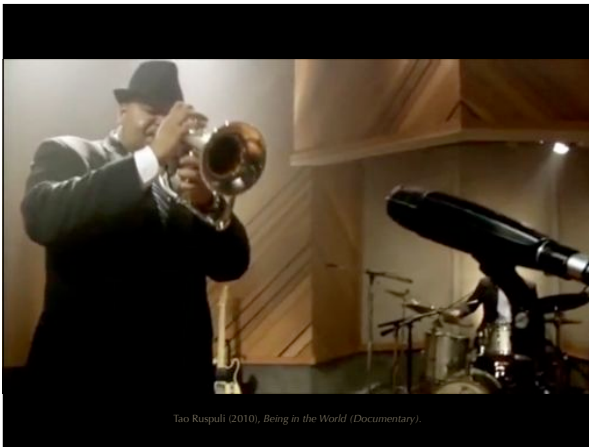


Dat geldt niet voor het werk van de programmeur: dat is onzichtbaar zo gauw het werkt.

werkwijze



De dialoog die de kunstenaar aangaat met de materie...



De jazz-muzikant die samenspeelt met de anderen verdwijnt in het grotere geheel.

Gerade in der großen Kunst [...] bleibt der Künstler gegenüber dem Werk etwas Gleichgültiges, fast wie ein im Schaffen *sich selbst vernichtender Durchgang* für den Hervorgang des Werkes.

Heidegger 2003[1950], p.26

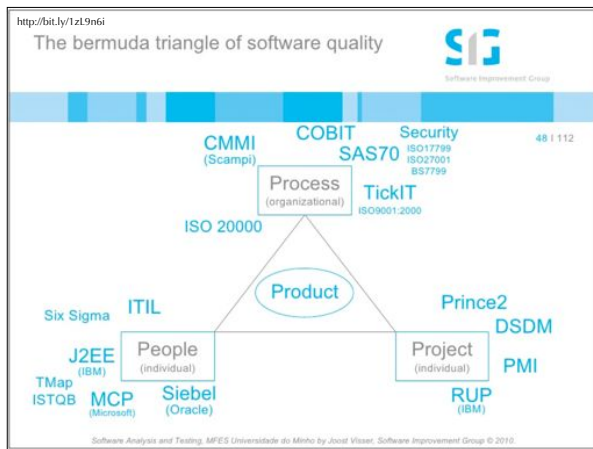
Mark Wildschut vertaalt *Durchgang* met *Medium*.

Het gaat ook in programmeren, in software engineering om het *bouwen* van dingen. Dan komen ze pas tevoorschijn.

```
/home/anton/prj/cpp/snake:
total used in directory 8 available 29813824
drwxr-xr-x  2 anton anton 4096 2009-11-28 15:48 .
drwxr-xr-x 72 anton anton 4096 2009-11-28 15:48 ..
```

-U:%%- snake All L5 (Dired by name)-----

<https://www.youtube.com/watch?v=GjZGFBGgKU>



conclusie

Het dagelijks werk van een programmeur
lijkt meer op dat van een kunstenaar dan
op dat van een ambachtsman.

Dit betekent dat we in het informatica-onderwijs meer
aandacht moeten besteden aan het creatieve proces.



In het onderwijs moeten we dus minder werken met standaard ingenieur-
achtige werkvormen ...



... en meer aandacht schenken aan het creatieve maakproces.

b.barnard@pl.hanze.nl

Referenties:

Cooper, A. (1999), *The Inmates are Running the Asylum. Why High-Tech Products Drive us Crazy and how to Restore the Sanity*. Indianapolis, IN: Sam's Publishing House.

Heidegger 2003[1950], *Holzwege*. Frankfurt aM: Vittorio Klostermann.

Juarez, U.M. (2010), *Beauty in Mathematics*. Proefschrift Rijksuniversiteit Groningen.

Martin, R.C. (2009), *Clean Code. A Handbook of Agile Software Craftsmanship*. Boston, MA: Pearson Education, Inc.

Nehamas, A. (2007), *Only a Promise of Happiness. The Place of Beauty in a World of Art*. Princeton UP.
