# Beauty and Code

Tom Brus

Modeling Value Group

*"Beauty is more important in computing than anywhere else in technology because software is so complicated. Beauty is the ultimate defence against complexity."*
(David Gelernter) [1]

## 1 Introduction

Let me take you on an expedition in search of the beauty in code. Although not everybody links code and beauty, it has always been a natural link for me. Apparently I am not the only one, considering the hefty battles that show up regularly regarding syntax, coding practices, and how things should be written down. I have some strong feelings in this area myself and fought some battles in the past. But syntax and the appearance of code on screen or paper is not the only beauty axis that can be identified, there is more which I also hope to illustrate below.

I have always looked on coding (or code creation) as a creative process and 'creative' encapsulates a drive for beauty for me, at least in some direction. I always searched for satisfaction by creating something that I can look back on with a certain satisfaction, some intricate sensation of beauty. The more beauty, the more satisfaction. The coding process is more worthwhile, more satisfying when more beauty is created.

When you start thinking about beauty in code you realize that there are many aspects of code and code-creation that can be beautiful. For instance: the resulting program can have a nice GUI; the source code can look elegant; the program can have a pleasingly simple structure.

So, to identify beautiful code and distinguish it from the opposite (ugly code? mediocre code?) we need to identify what beautiful could mean for code. We need reasons to label code as beautiful or not. Only then we can find the real beauty.

But is a quest for beauty a good guidance for high quality code? To answer that question we should investigate what 'quality' exactly means for code and that is beyond this essay. For me personally, 'high quality' and 'beautiful code' go together like copy and paste. Whenever there seems to be only one you need to step back and rethink.

P. Achten and P. Koopman (Eds.): Plasmeijer Festschrift, LNCS 8106, pp. 8–17, 2013.

My plan in this essay is to first look at what beauty is. Then I look into what that specifically means for code. Finally I will look at different coding languages as well as programming paradigms and how they can be related to beauty.

In this essay I do not claim to be complete in any dimension nor do I intend to identify universal truths. It is not academic or scientific, and although some experience with programming helps to enjoy this essay, it is not technical either. I hope to shine a small light from a personal angle on what the art and beauty of programming is. I am convinced that it is an art and that it should remain an art [2]. In all humbleness I think I am old enough to look back and talk about the past. I even enjoy doing that, at least occasionally.

## 2 Some Historical Notes

When I started programming around 1976 (in 8080 assembler) I did not have any clear notion of beauty in mind. Getting the thing to do what I wanted it to do was my goal. I can remember, however, that I did rearrange code fragments only because it would look better or maybe feel better, even if there was no technical reason to do so. In hindsight I am tempted to conclude that I was already trying to create beauty, crude as it was. I also remember that code size really mattered and that cramping the same functionality into a smaller code size was a satisfying and rewarding process. You could also win points among peers for a yet smaller footprint. This had little to do with beauty I think, although some see beauty in small waists or small feet. Not me, so that must have just been a practical issue. Nowadays code size is not a real issue anymore and we tend to even spend more space to avoid riddles in maintenance or even for laziness. The quest for time on the other hand has not degraded that much. Machines have gotten ridiculously faster but we also seem to be more hungry for cycles. So squashing the last drop out of the available cycles is at times still a rewarding exercise.

Later I was charmed by the first Macintosh computer because of its elegance. Real fonts, a super intuitive mouse device, a wealth of GUI primitives and last but not least a fanless design. At that time I earned my first $5 in 1987 with ‘m2beauty’, a public domain code beautifier for Modula-2 (this was by the way also the only donation for this project). When researching for this essay I was surprisingly still able to find it on the internet. I guess we need to wait for the first retro-Mac-on-iPhone-emulator to be able to run it again (and find a Modula-2 programmer to use it, which might even prove to be more difficult).

When making the first Clean compiler, around 1988, based on graph reduction for a VAX, I can remember that performance was the big issue. This was mainly due to the fact that functional coding was regarded by the world as an academic exercise that was good for proving theorems but was not worthwhile for everyday programming. We, Rinus Plasmeijer and his team, tried to convince the world of the opposite and at that time we needed the performance to support our argument. I believe we gave Clean a good head start in that lane. The position of functional programming has changed since that time. Functional as well as declarative programming has become more mainstream. Still it has not reached

the acceptance levels that C had at the time we designed Clean. It is actually amazing how much ground is still covered by C at the present time. I remember that in the 80's and 90's I was implicitly convinced that new programming languages and paradigms would follow each other like assembler was followed by C which was followed by Pascal/Modula. I see a far more scattered field now. Java has takes a big chunk (and rightfully so) but is still almost a par with C according to the TIOBE index for April 2013 [3]. The index for February 2013 mentions Clean in the number 51...100 popularity range of the TIOBE index, but has unfortunately dropped below 100 in the recent April issue. Functional languages in general only have a mere 3.1% popularity score in the TIOBE index of April 2013. I must add that the methods and accuracy of this index can rightly be questioned, but it is at least an indication.

Going back to beauty, one could state that there is not enough generally accepted beauty in functional languages to make them generally used. We can of course also be convinced that the intrinsic beauty is unrecognized by the wider audience. My personal opinion on this matter is that there is a lot more involved in getting a language accepted than just beauty. There is a complex collection of reasons why a language is picked up. Apart from all sorts of technical qualities, sheer luck is one of those factors, or in other words being the right language at the right time. Being fashionable is another (this hooks into beauty by the way).

## 3 What Is Beauty

What does beauty mean when we are not referring to code? What is beauty in our day to day life? There are aspects of *balance*, *harmony*, *feelings of well being*, *attraction*, *satisfaction*. In our western society there is, without doubt, a special relationship between beauty and humans, ourselves. This is generally interpreted as outer beauty. Generally, but not solely. There is also inner beauty, which is harder to get a grip on and needs time and experience to recognize.

I think you can identify those two beauty aspects in code as well. There is outer beauty, the arrangement of the lines, how pleasing it is to look at the code on screen or paper. There is also inner beauty: how well does the code describe and define the solution at hand. As with beauty in humans, the outer beauty aspects can be tweaked to a certain extent without changing the essence of the code. This is where code-beautifiers play a role, they are the beauty parlours for software engineers. Nothing much is changed, it just looks better. If we go a step further we can do some plastic surgery. I would like to compare that to refactorings of software: you are really changing your code, but the meaning of the code is not changed. You make semantic changes but they result in different but equivalent code. When we look at inner beauty it is harder to make make changes for the better. I would like to compare changes on that level to meditation for humans: you really have to sit down and think about algorithms and solutions and most of the time there are moments of insight where the essence of your code is changed. These kind of changes are harder, and as of yet impossible, to automate. They need human creativity and perseverance to be accomplished. As with meditation

and other inner beauty changing processes they are hard to accomplish in code. First of all they are very hard to just do, but secondly they are hard to start because it involves rethinking your solution and possibly throwing away your initial one. Depending on the time you spent already it might seem a waste. The endeavour of rethinking everything from the ground might also seem a waste because you are (inherently) not sure if you are going to find a better solution. This is where our wonderful mind jumps in: we need to trust our instinct. If our instinct tells us that there should be a better (more beautiful) solution then it is worthwhile investigating. Finding it in the end is always uncertain and that is the beauty of the endeavour.

### 3.1 Outer Beauty

I have always been implicitly convinced that outer beauty is not the most important thing in life. On the other hand it is not completely unimportant either. I would like to introduce what I call 'modest beauty': the beauty that does not shout out: "I am beautiful". It is the beauty that needs to be discovered. You will have to find balance and the harmony in things and discover the true beauty that caused that feeling. I have been a lifelong fan of Apple products because of this. There have never been stickers on Mac's that shout out the GHz-s and other mega-multi-super qualifications. The thing is just there. You need to discover the beauty of it, feel the balance and use it with a smile.

This modest beauty concept equally applies to humans. I find little beauty in magazines with photoshopped people, they have beauty written all over them but I do not seem to find it. Real people have imperfections and if they have learned to live with those imperfections and convey balance and harmony there is a high chance they are beautiful in a modest way.

For objects as well as people this modest beauty does not come for free. It probably takes more effort to design a Mac laptop then an average windows laptop. Equivalently, getting in balance with your imperfections and be who you are is not a trivial exercise either. In my opinion modest beauty is real beauty.

Does this relate to code? Maybe it does, maybe not. The outer beauty aspects of code that I value are mainly based on making the meaning more easy to grasp. For instance: I like columnization. The visual appearance is more balanced, there is more harmony, maybe more beauty.

Take this arbitrary snippet for example:

```
private int cacheLength = ApnsConnection.DEFAULT_CACHE_LENGTH;
private boolean autoAdjustCacheLangth = true;
private ExecutorService executor = null;
private ReconnectPolicy reconnectPolicy = ReconnectPolicy.newObject();
private boolean isQueued = false;
private ApnsDelegate delegate = ApnsDelegate.EMPTY;
private Proxy proxy;
private boolean errorDetection = true;
```

I find more balance and harmony in this representation:

```
private int             cacheLength           = ApnsConnection.DEFAULT_LENGTH;
private boolean         autoAdjustCacheLangth = true;
private ExecutorService executor              = null;
private ReconnectPolicy reconnectPolicy       = ReconnectPolicy.newObject();
private boolean         isQueued              = false;
private ApnsDelegate    delegate              = ApnsDelegate.EMPTY;
private Proxy           proxy                 = null;
private boolean         errorDetection        = true;
```

I also like to see uniformity in control structures and how they are written down (i.e. write all curly brackets and write them always at the same position). Another arbitrary snippet:

```
if (minutes) return minutes.minutes();
else if (hours) return hours.hours();
else return 0;
```

for which I would prefer this make-up:

```
if (minutes) {
    return minutes.minutes();
} else if (hours) {
    return hours.hours();
} else {
    return 0;
}
```

This example actually proves my earlier statement about beauty being personal, because the last example was taken from [4], where the author is talking about the beauty of braces being absent. The absence of braces is more 'zen' in his opinion while in my opinion it gives a more uniform structure. Two different views on beauty.

In general I think terseness can not clearly be related to beauty. In my opinion it is obvious that very verbose code or a programming language that requires it is not very beautiful, because the verbosity hides the essence of the solution. In contrast, a very terse program or language hides the essence in riddles. When I was first exposed to APL I was charmed by the beauty of the terse expressions that could express everything but the kitchen sink. Later on I realized that it was actually a way of hiding meaning for everybody but the experienced APL programmer. Take for instance this Sudoku solver in APL presented in [5]:

```
Sudoku←{
   box  ← {ω ≠ ω/ω  ωριω∗2}
   rcb  ← {(ιω), ¨box⊃ ω∗0.5}
   cmap ← {⊂[ι 2] 1∈¨ ω°.=ω}
   CMAP ← cmap rcb ρω
   at   ← {ω+α×(ιρω)∈⊂ αα}
   avl  ← {(ι ⊃ ρω)˜ω× ⊃ α⌷CMAP}
   emt  ← {(,ω=0)/,ιρω}
   pvec ← {(α avl ω) (α at)¨ ⊂ ω}
   pvex ← {⊃,/α°pvec¨ω}
   svec ← {⊃pvex/(emt ω),⊂⊂ ω}
   svec ω
}
```

Very terse, very cryptic and hardly beautiful, at least for me. Beauty, again, turns out a matter of taste since the solution is literally presented as *"... the following more beautiful form..."* in the referenced document. You can watch a YouTube movie [6] if you like a detailed explanation of this solution.

All these outer beauty aspects give me a more balanced and uniform visual experience which in turn makes the program easier to interpret and understand. For me, this is the essence of beauty in programming: to help relaying the meaning of your program while still defining it in an executable (sometimes even provable) way.

It is not only important to relay the meaning of what you scribbled down to other people, but also to yourself. Many times I took code I had written years ago (voluntarily or not) and tried to make sense of it. Sometimes it was hard, more times it was harder.

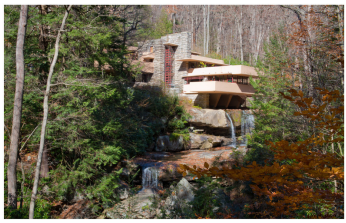
I would like to add here that this last aspect of beauty in code (better understandable code is also more beautiful) does not necessarily coincide with our perception of beauty in the real world. A beautiful woman is not always the easiest to understand. A certain level of mystery can make a woman even more beautiful. I guess women and code are not completely comparable after all.

### 3.2 Decay over Time

As in everyday live, beauty seems to evaporate over time for code as well. The code you can still remember you were proud of ten years ago is not so beautiful anymore. For our human body this is mostly due to actual changes happening over time. Code does not change when you do not touch it so it must be a change in our mind that makes it less beautiful.

This is comparable to fashion: what was beautiful in the 70's does not get that qualification anymore, at least not from me. As with code, the material did not changed but we did.

There are, of course, timeless designs that keep their beauty over time. The designs of Frank Lloyd Wright are definitely in that category for me. Whether those designs are genuinely timeless or merely longer living, only time will tell.



The Sagrada Familia in Barcelona is a perfect example of beauty in a potentially timeless setting. Generations have been building it now and still it feels as one beautiful piece of art. We can only dream of IT projects of comparable magnitude to result in such splendor. (image: Gubin Yury / Shutterstock.com)

I always experience harmony, balance and serenity when looking at zen gardens. Will code ever look like this?

If you have kids of an older age you will probably recognize this: when your child was born you genuinely and objectively felt that it was by far the most beautiful child in the world. When seeing his or her baby photos after many years you might not be so sure anymore. I can see a parallel in code beauty here... When writing a program you really think it is cool and beautiful. On a higher level: when designing a programming language you also think it is the most beautiful language you have ever seen. After years of nurturing you find yourself with a product or language that is well equipped for life but also has its scars. When looking back at the first designs and implementations you might be a little furtively ashamed and at the same time touched by reminiscences.

What I aim for from the start is beauty, although it might be only raw beauty at first. Sometimes this raw beauty will shine later, sometimes it drowns. I must admit that at least some of my older code generates more sense of shame

than arousal of beauty now. Nevertheless my goal is to write code (or design languages) that I can look back on with satisfaction, even after many years. This is the reason why I am hesitant to engulf myself in the latest hype, the fashion areas of our industry. At the same time I regard it as a challenge to pick out the hypes with long term potential. An intriguing and endless balancing game.

### 3.3 Inner Beauty

I discussed some aspects of inner beauty above already. When a program can be easily understood by reading, there is beauty. A program should not be written with the executing computer in mind but with the human reader in mind, he or she is your peer who needs to understand what you are concocting. Beauty is not something that a computer requires to execute a program. I can still remember one of my first professors at the university, Kees Koster, teach us to program top down in Algol 68. He advocated that subroutines or functions are also valuable if only used once. This was contradicting my feeling for machine efficiency but made sense in a peer to peer setting where you wanted to explain what is happening. We were stimulated to use spaces inside ridiculously long identifiers. This made you concentrate on describing the solution more than programming it. To get a feeling, consider this imaginary snippet (I do not remember the exact syntax of Algol 68):

```
PROC build a house according to the specifications of the designer
BEGIN
    remove any buildings from the land if they are present;
    build the house on cleared land;
END

PROC build the house on cleared land
BEGIN
    build a strong foundation;
    build the all walls for the whole house;
    put the roof on the house;
END

PROC make the foundation
BEGIN
    dig a hole for the foundation;
    pour concrete to make the foundation;
    pour concrete for the ground floor;
END

PROC build all the walls for the whole house
BEGIN
    WHILE the walls are not high enough for the average person in this area DO
        put another row of bricks on all the walls;
END
```

In hindsight it was a bit over the top for production environments but was definitely instructive in the educational setting. Without knowing it we were aiming for inner beauty: explainable program structures that were understandable and clearly identifiable.

Conveying meaning is the essence and beauty of code.

One of the essential spoilers of understandable and clear code is the inevitable time sequential specification in imperative languages. It is one of those little critters in programming we could happily do without. It always spoils things. There is always a sequence of events that you did not take into account. In catering for these rare sequences you add code and then more code, you add complexity and make it less understandable, less beautiful, often even ugly. An additional problem with sequential specifications in programming languages is that it is only essential at some points. At many points in your program it is not important and therefore an over specification. The main frustration comes from the fact that it can not be specified where sequential execution is essential and where it is not. It is then very hard, if not impossible, for humans as well as computers to separate the two. We need programming languages that do not require time sequential specifications to make truly beautiful code. We need to abstract away from time to encounter real beauty. This perhaps explains my preference for declarative and functional programming where sequentiality is not explicitly specified. When you free yourself from this over-specification there is more beauty.

So why then is functional and declarative programming used so little ($< 5\%$ currently and going down!). Well, it has taken Apple decades to become the biggest company on earth. Why was the beauty not picked up earlier? Maybe beauty is not the thing that drives the avarage human being, maybe it is because sheer beauty is not enough, maybe you need a killer-app for wide acceptance. I do not have the answers. What I know is that you need to follow your instincts and my instinct tells me to follow the trail of functional and declarative, because beauty can be found there. Maybe I will live long enough to see it widely accepted, it would make me happy and even more happy to play a modest role in that.

*When I am working on a problem I never think about beauty.*
*I think only how to solve the problem.*
*But when I have finished, if the solution is not beautiful, I know it is wrong.*
(Richard Buckminster Fuller) [7]

## References

1. Gelernter, D.: Machine Beauty: Elegance And The Heart Of Technology. Basic Books (1998)
2. Knuth, D.: Computer programming as an art. Communications of the ACM 17, 667–673 (1974), ACM Turing Award Lecture

3. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
4. http://tobyho.com/2009/02/01/programming-without-braces/
5. http://dfns.dyalog.com/n_sudoku.htm
6. http://www.youtube.com/watch?v=DmT80OseAGs
7. Buckminster Fuller, R.: US architect and engineer (1895 - 1983), quote taken from http://simple.wikiquote.org/wiki/Richard_Buckminster_Fuller