Code Beauty

Prof. Fabio Kon

Department of Computer Science
Institute of Mathematics and Statistics
University of São Paulo

SugarLoafPLoP'2010 - 24/9/2010
Salvador, Brazil

Agenda

- Definition of Beauty
- Is Code Beauty important for Software Development?
- What is Code Beauty

Beauty - Ancient View

- pre-socratic (e.g., Pythagoras): strong connection among Beauty and Mathematics; objects with proportions following the golden ratio are more attractive to the human brain
- *two quantities are in the golden ratio if the ratio of the sum of the quantities to the larger quantity is equal to the ratio of the larger quantity to the smaller one. The golden ratio is an irrational mathematical constant, approximately 1.6180339887*
- greek architecture is based on symmetry and on the golden ratio

Beauty - Romantic View

- works do not need to be harmoniously proportioned and constitute a well-rounded whole to be beautiful; on the contrary, dissonance and fragmentation is beautiful
- more precisely: dissonance and fragmentation allude to a harmony and wholeness that is ***not*** in the work

Beauty - Modernist View

- architect Mies van der Rohe: "*Less is More*"

  ~ 1940

- designer Buckminster Fuller (geodesic dome) adopted the engineer's goal of "*Doing more with less*"
- *writer Antoine de Saint Exupéry (1939) "perfection is attained not when there is nothing more to add, but when there is nothing more to remove"*

Beauty: Post-modernist view

- architect Robert Venturi: *“Less is a Bore”, let's do whatever people like* the greek and modernist views of beauty provide us with a good basis for approaching Beauty in Science and Technology

Science

make everything as simple as possible, but not simpler

Technology

Conceptual integrity

- the architect or team of architects should develop an idea of what the system should do and make sure this vision is understood by the rest of the team
- to ensure a user-friendly system, a system may deliberately provide fewer features than it is capable of

Beauty dictionary definition

the quality that gives pleasure to the mind or senses and is associated with such properties as

- *harmony of form or color,*
- *excellence of artistry,*
- *truthfulness,*
- *and originality.*

The American Heritage Dictionary of the English Language, 4th Edition, 2000.

Does it matter?

Why are we talking about Code Beauty on a “serious” Software Conference???

- personally, I have two reasons...

- there has been a strong movement towards managing software development projects as any other production process/system
- many cases of managers that have never written a single line of code
- is it possible to have a good *Chef de Cuisine* that has never cooked in his life?

- scientists/engineers/practitioners sometimes tend to overestimate the power of science and mathematics

- can science and engineering provide all what's needed to explain life? and software development?

What is Software Development?

– Modeling	(Jacobsen)
– Engineering	(Meyer)
– Discipline	(Humphreys)
– Poetry	(Cockburn)
– Craft	(Knuth)
– Art	(Gabriel)

(from from Alistair Cockburn)

- Common mistake: look at software as only one of the above items.

- our goal here, is to bring attention to a few important aspects of software development that are neglected in some software engineering communities

- we'll look at the most important deliverable on a software project:

The Code

Beauty is fundamental

vinicius de morais

Beautiful code

- brings pleasure to the reader
- makes the writer happy
- makes working in groups fun

This leads to

- fewer bugs
- maintainability
- team productivity
- in other words, Quality

In Software Development,

Beauty

Leads to

Quality

What is Beauty in Software Development?

sources:

- Robert C. Martin. *Clean Code - A Handbook of Agile Software Craftsmanship*. Prentice Hall. 2008.
- Andy Oram and Greg Wilson. *Beautiful Code*. O'Reilly. 2007.
- beaute(code) art exhibit by Bob Hanmer, Karen Hanmer, and Andrea Polli.
- exchange with experts

Rebecca Wirfs-Brock's code beauty

- *it is a common recognition that these idioms and those structures and those ways of doing things are of value*

- *it is important that a project or a team have the same sense of aesthetics or there will be clashes of will*

- *keeping of every step in a method at the same level of abstraction/intention; it reads much more like prose that way*

- *characteristics that are "normally" present in beautiful code are balance, effectiveness, expressiveness, and profoundly doing well what it was designed to do.*

Rebecca's example of a beautiful code

• the Smalltalk class collection

• provide all the built in behavior by requiring subclasses to implement three methods:

•**`add: anObject`**
•**`remove: anObject ifAbsent: exceptionBlock`**
•**`do: aBlock`**

• *methods in the abstract class collection to add, remove, check for emptiness, inclusion, occurrences, and to step through a collection in different ways accumulating values… are all implemented use these abstract building blocks or invoking other defined behaviors that depend on these implementation; elegant, beautiful reuse!*

Joe Yoder's code beauty

- *I think it is important to most of us who live there; it is like cleaning up our living area, having clean code or beautiful code can make life much better*

- *however let's face it, BBoM still thrive and exist and are successful*

- *it matters, but is not necessary and sufficient for successful software projects. If so, then Clean Code would become the norm*

- *those of us who care strive for it because we believe it is the right thing to do and it will help us in the long run...but certain forces are in play that can make clean code or beautiful code a goal that is very hard to achieve*

Joe's example of a beautiful code

• also the Smalltalk class collection
• *you see a certain beauty and elegance in the system as it was easy to understand and extend*

• (it's also my favorite piece of code :-)

• *design patterns can help with this as well; if a system is clean and well designed by implementing some of the best known principles and practices one can see certain beauty in the system*

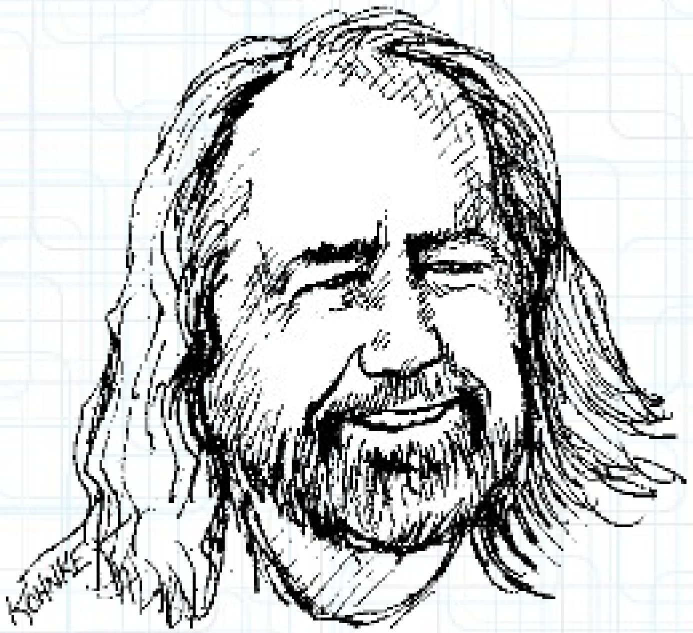
What is Clean Code?

Bjarne Stroustrup

Inventor of C++

“I like my code to be **elegant** and **efficient**. The logic should be **straighforward** to make it hard for bugs to hide, the **dependencies minimal to ease maintenance**, **error handling complete** according to an articulated strategy, and **performance close to optimal** so as not to tempt people to make the code messy with unprincipled optimizations. **Clean code does one thing well.**”

What is Clean Code?

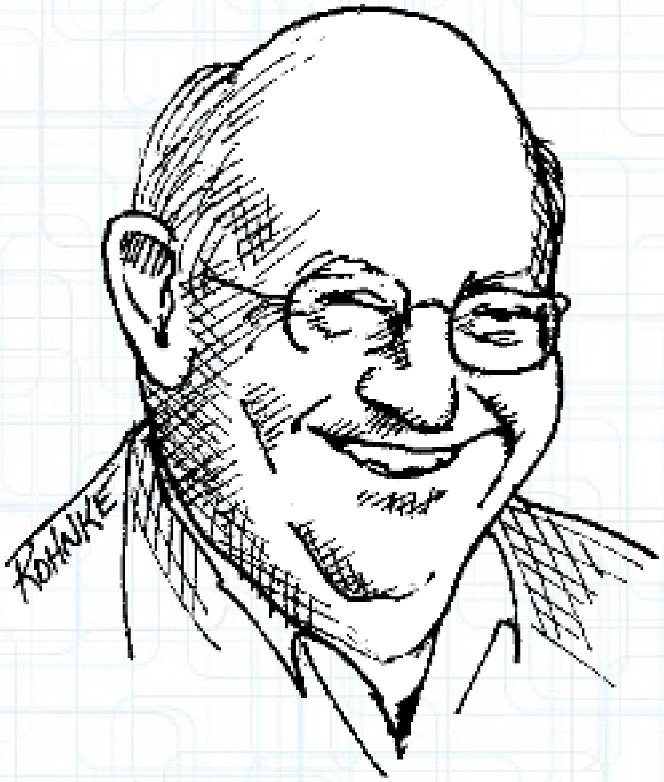


“Clean code is **simple** and **direct**. Clean code **reads like well-written prose**. Clean code never obscures the designer's intent but rather is full of crisp [clearly defined] abstractions and **straighforward** lines of control.”

Grady Booch

Author of *Object Oriented Analysis and Design with Applications*

What is Clean Code?



Dave Thomas
Founder of OTI, godfather of the Eclipse Strategy

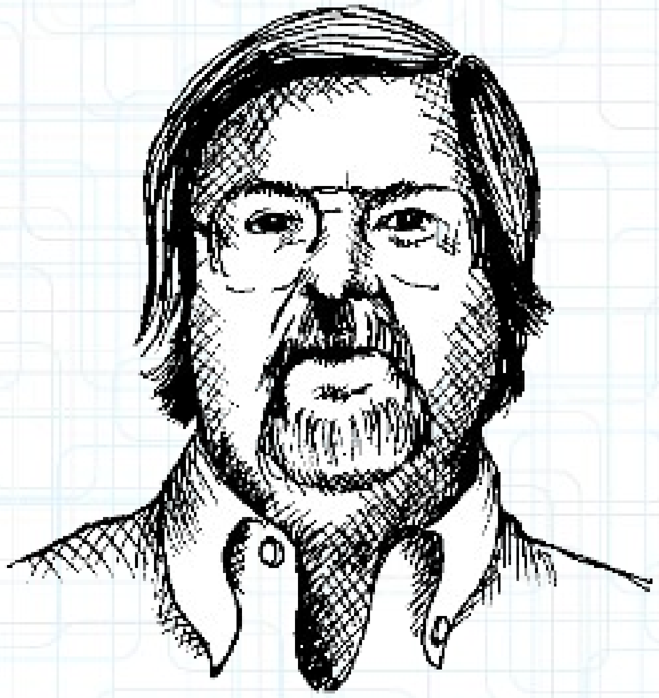
"**Clean code can be read**, and enhanced by a developer other than its original author. **It has unit and acceptance tests**. It has **meaningful names**. It provides **one way** rather than many ways for doing one thing. It has **minimal dependencies**, which are explicitly defined, and provides a clear and **minimal API**. Code should be **literate** since depending on the language, not all necessary information can be expressed clearly in code alone."

What is Clean Code?

Michael Feathers
Author of *Working Effectively With Legacy Code*

"I could list all of the qualities that I notice in clean code, but there is one overarching quality that leads to all of them. **Clean code always looks it was written by someone who cares**. **There is nothing obvious that you can do to make it better**. All of those things were thought about by the code's author, and if you try to imagine improvements, you're led back to where you are, sitting in appreciation of the code someone left for you – **code left by someone who cares deeply about the craft**."

What is Clean Code?



Ron Jeffries
Author of *Extreme Programming Installed*

“In recent years I begin, and nearly end, with Beck's rules of simple code. In priority order, simple code:

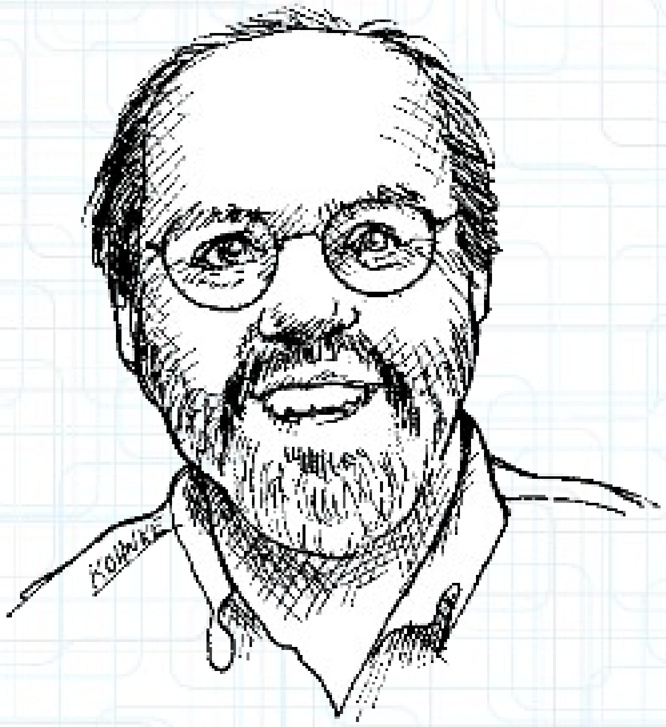
· **Runs all tests**

· **Contains no duplication**

· **Expresses all the design ideas** that are in the system

· **Minimizes the number of entities** such as classes, methods, functions, and the like.”

What is Clean Code?



Ward Cunningham

Inventor of Wiki, Fit and much more

"Godfather of all those who care about code"

You know you are working on clean code when **each routine you read turns out to be pretty much what you expected**. You can call it **beautiful** code when the codes also **makes it look like the language was made for the problem**."

What is Clean Code?

Simple

Efficient

Without obvious improvements

Straightforward

Expressive

Contains no duplications

Turns out to be what you expected

Runs all tests

Full of meaning

Literal

Reads well

Beautiful: when the language was made for the problem

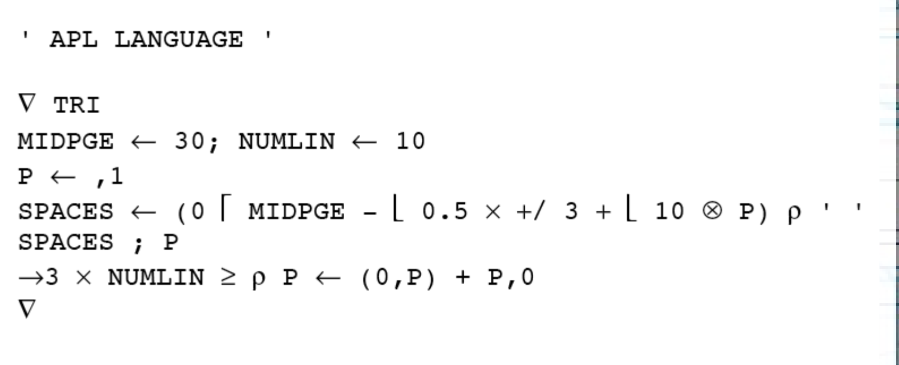
Minimal

Written by someone who cares

language may influence

```
/*  C Language  */
#include <stdio.h>
#define NumLines 10                    /* number of lines */
main(argc,argv) int argc; char **argv;
{
    int tri[NumLines][NumLines],  /* triangle */
    r,c,i;                        /* row, column, misc index */

    /* initialize array */
    for (r=0;r<NumLines;r++) {
        tri[r][0]=1;
        for (c=1;c<NumLines;c++) {
            tri[r][c]=0;
    };
    /* generate triangle */
    tri[0][0]=1;
    for (r=1;r<NumLines;r++){
        for (c=1;c<NumLines;c++) {
            tri[r][c]=tri[r-1][c] + tri[r-1][c-1];
        }
    };
    /* print triangle */
    for (r=0;r<NumLines;r++){
          for (i=0;i < ((NumLines-r-1)/2);i++) printf("    ");
          if (r%2 == 0) printf("  ");
        for (c=0;c<NumLines;c++){
            if (tri[r][c] != 0) printf("%3d ",tri[r][c]);
        };
        printf("\n");
    }
}
```



```
' APL LANGUAGE '

∇ TRI
MIDPGE ← 30; NUMLIN ← 10
P ← ,1
SPACES ← (0 ⌈ MIDPGE - ⌊ 0.5 × +/ 3 + ⌊ 10 ⊗ P) ρ ' '
SPACES ; P
→3 × NUMLIN ≥ ρ P ← (0,P) + P,0
∇
```

source: http://www.andreapolli.com/beaute%28code%29/good_bad_ugly/pascal.html let's get a little more technical

Meaningful Names

Meaningful Names

Names are vital!

- Code is basically names and reserved words
- Choosing good names takes time but saves more than it takes
- Names should be expressive and should answer questions

Meaningful Names

Example

```
public List<int[]> getThem() {
   List<int[]> list1 = new ArrayList<int[]>();
   for (int[] x : theList)
      if(x[0] == 4)
         list1.add(x);
   return list1;
}
```

Meaningful Names

Example

```
public List<int[]> getThem() {
    List<int[]> list1 = new ArrayList<int[]>();
    for (int[] x : theList)
        if(x[0] == 4)
            list1.add(x);
    return list1;
}
```

Many doubts arise...

1. What does this method get?
2. What kinds of things are in theList?
3. What is the importance of the zeroth position?
4. What is the significance of the value 4?

Meaningful Names

Example

```
public List<int[]> getThem() {
    List<int[]> list1 = new ArrayList<int[]>();
    for (int[] x : theList)
        if(x[0] == 4)
            list1.add(x);
    return list1;
}
```

What about this code?

```
public List<int[]> getFlaggedCells() {
    List<int[]> flaggedCells = new ArrayList<int[]>();
    for (int[] cell : gameBoard)
        if(cell[STATUS_VALUE] == FLAGGED)
            flaggedCells.add(cell);
    return flaggedCells;
}
```

Meaningful Names

Example

```
public List<int[]> getFlaggedCells() {
    List<int[]> flaggedCells = new ArrayList<int[]>();
    for (int[] cell : gameBoard)
        if(cell[STATUS_VALUE] == FLAGGED)
            flaggedCells.add(cell);
    return flaggedCells;
}
```

Problem
solved!

1. What does this method get? It gets all flagged cells!

2. What kinds of things are in theList? theList is a gameBoard filled with cells!

3. What is the importance of the zeroth position? That's the Status Value!

4. What is the significance of the value 4? It means it is flagged!

Meaningful Names

Example

```
public List<int[]> getFlaggedCells() {
    List<int[]> flaggedCells = new ArrayList<int[]>();
    for (int[] cell : gameBoard)
        if(cell[STATUS_VALUE] == FLAGGED)
            flaggedCells.add(cell);
    return flaggedCells;
}
```

Going further...

```
public List<Cell> getFlaggedCells() {
    List<Cell> flaggedCells = new ArrayList<Cell>();
    for (Cell cell : gameBoard)
        if(cell.isFlagged())
            flaggedCells.add(cell);
    return flaggedCells;
}
```

This is pretty much what you expected!

Changes should be easy!

To make changes, we need to understand the code!

- **Use Readable names**
  - XYZControllerHandlingOfStrings != XYZControllerStorageOfStrings
- **Use Searchable names**
- **Use the Language Standards**
- **Use Solution Domain names**
  - Use pattern and algorithm names, math terms, …
- **Use Problem Domain names**
- **Don't confuse the reader**
  - Use the ”One Word Per Concept” rule
  - Don't use jokes, mind mappings, hungarian notation, ...

Conclusion

• it is easy to say that names should reveal intent. What we want to impress upon you is that we are *serious* about this

• short names are generally better than longer ones, so long as they are clear (do not promote obscurity to save a couple of keystrokes)

• **if you find a bad name, change it now!**

Functions

Functions

Functions should be small!

- **functions should have few lines**
  - each of them should be obvious and easy to understand

- **functions should not hold nested structures**
  - if, while, else blocks should be straightforward
    (probably a function call)
  - the conditional should probably be a function call that encapsulates it

Functions

One Thing!

Functions should do one thing.

They should do it well.

They should do it only.

- **functions that do one thing can't be divided into sections**

- **two ways to identify whether a function does One Thing**
  - if a function does only those steps that are one level below the stated name of the functions, then the functions is doing one thing
  - If you can't extract another function from it with a name that is not merely a restatement of its implementation, then it's doing one thing

Example

```
public void pay() {
    for (Employee e : employees) {
        if (e.isPayday()) {
            Money pay = e.calculatePay();
            e.deliverPay(pay);
        }
    }
}
```

It does more than one thing...

1. it loops over all the employees

2. checks to see whether each employee ought to be payed

3. pays the employee

Refactored Example

```
public void pay() {
    for (Employee e : employees)
        payIfNecessary(e);
}
```

It just iterates over the employees

```
private void payIfNecessary(Employee e) {
    if (e.isPayday())
        calculateAndDeliverPay();
}
```

Checks whether an employee ought to be paid

```
private void
calculateAndDeliverPay(Employee e) {
    Money pay = e.calculatePay();
    e.deliverPay(pay);
}
```

Pays the employee

is this exaggerated?

Functions

One level of Abstraction

- **statements within a function should be all in the same level**

- **mixing levels is confusing**
  - once details are mixed with essential concepts, more and more details tend to accrete within the functions
  - it's the first step towards the creation of big functions!

- **the Stepdown rule**
  - we want code to read like a top-down narrative

Functions

Function arguments

- **functions should minimize the number of arguments**
  - arguments are hard from a testing point of view
  - too many arguments = the function does more than one thing
  - too many arguments = the function is used in many different ways

- **don't use flag arguments**
  - it loudly proclaims that the functions is doing more than one thing

Functions

Last but not least

- **functions should not have side effects**
  - they usually create temporal coupling
  - it should create a effect on the object **or** return something

- **Don't Repeat Yourself (DRY)**
  - duplication may be the root of all evil in software

functions should be short, well named and nicely organized

Comments

”Nothing can be quite so helpful as a well-placed comment. Nothing can clutter up a module than frivolous dogmatic comments. Nothing can be quite so damaging as an old crufty comment that propagates lies and misinformation.”

Comments

The problem with comments

- **comments are, at best, necessary evil**
  - the proper use of comments is to compensate for our failure to express ourself in code; note that I used the word failure; I meant it

- **they lie!**
  - programmers can't realistically maintain them
  - comments don't always follow the code changes
  - they require a maintenance effort that takes time
  - truth can only be found in one place: the code

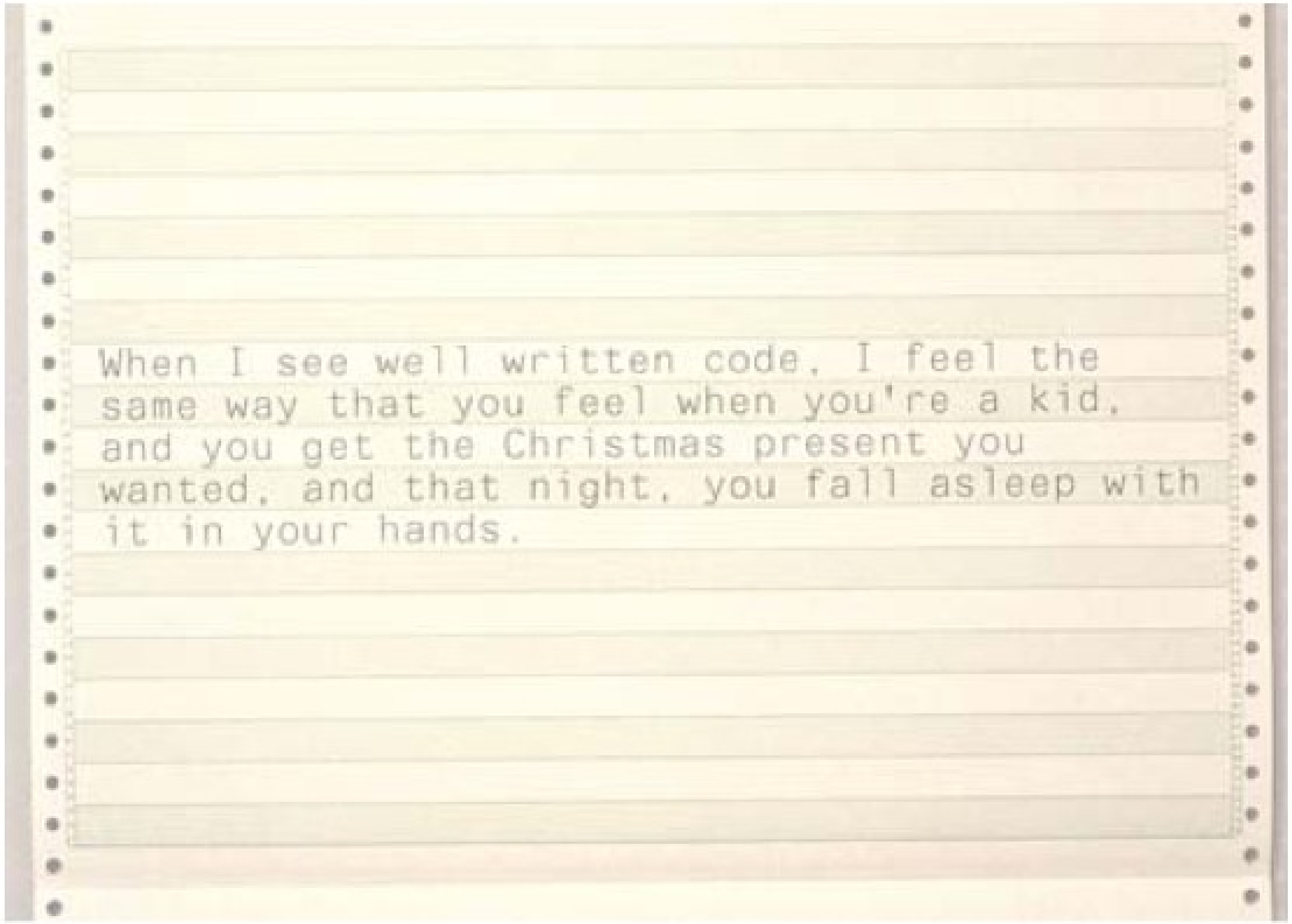
Comments

Code is the only truth

Good Comments

- **legal comments**
- **informative comments**
  - Commenting regular expressions can be quite useful
- **explanation of intent and clarifications**
  - some decisions aren't implementation decisions
  - we have to use libraries that aren't so expressive
- **Amplification**
  - explain how important an element is
- **TODO Comments and Javadocs in Public API**

Title: ***I Remember My First (installation version)***



```
When I see well written code, I feel the
same way that you feel when you're a kid,
and you get the Christmas present you
wanted, and that night, you fall asleep with
it in your hands.
```

I Remember My First (installation version) | 2002

Pigment inkjet prints on green bar computer paper

7 panels at 77 x 15" (installation 77 x 140")

Quotes from interviews with software engineers exploring how they became interested in working with computers and what they still find compelling about software, hardware and the act of programming.

why should we write beautiful code?

to feel good with ourselves

Bibliography

- Wikipedia definition for Beauty, Golden Ratio, Minimalism. 2010.
- Fred Brooks. *The Mythical Man-Month: Essays on Software Engineering*. 1975.
- Dorthe Jørgensen. *The Metamorphosis of Beauty*. 2002.
- The American Heritage Dictionary of the English Language, 4th Edition, 2000.

Bibliography

- Robert C. Martin. *Clean Code - A Handbook of Agile Software Craftsmanship*. Prentice Hall. 2008.
- Andy Oram and Greg Wilson. *Beautiful Code*. O'Reilly. 2007.
- beaute(code) art exhibit by Bob Hanmer, Karen Hanmer, and Andrea Polli. Some material available at http://karenhanmer.com/gallery/?gallery=beautecode and http://www.andreapolli.com/beaute(code)
- Interviews with experts
- slides by João Machini de Miranda - IME/USP