Technische Universität München

Clean Code

Tutorial for Advanced Programming

Friedrich Menhorn

December 22, 2015

Contents

1. Introduction

2. Clean Code
   2.1 Code Refactoring Example
   2.2 Naming
   2.3 Methods
   2.4 General principles

3. Famous software bugs

4. References

Smart people about clean code

Bjarne Stroustrop, inventor of C++
"I like my code to be elegant and efficient. The logic should be straightforward [...], the dependencies minimal [...], error handling complete [...], and performance close to optimal [...]. Clean code does one thing well."

Ward Cunningham, inventor of Wiki
"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 code also makes it look like the language was made for the problem

Micheal Feathers, *Working Effectivly 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 like it was written by someone who cares.

Contents

1. Introduction

2. Clean Code

2.1 Code Refactoring Example

2.2 Naming

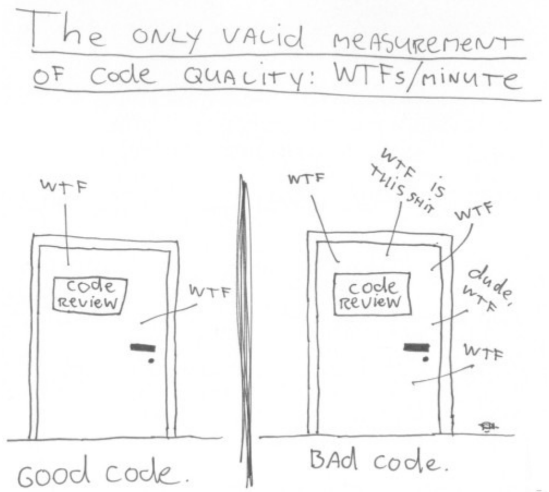
2.3 Methods

2.4 General principles

3. Famous software bugs

4. References

Think of your fellow coders



(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics

Code Refactoring

What is this method doing?

```
//This Functions opens text file , encodes the text and writes it to new file
static void bestFunctionEver(string fn){
    string lineArray;
    ifstream in(fn.c_str());
    ofstream out(("encoded"+fn).c_str());
    if (in)
    {
        while (getline( in, lineArray ))
        {
            for (int i=0;i<lineArray.length();i++){
                 if(lineArray[i]<='Z' && lineArray[i]>='A')
                    lineArray[i] = lineArray[i]-('Z'-'z');
             }

            int offset = 6;
            int nKey = offset%26;
            char ch;
            for (int i = 0; i < lineArray.length(); i++) {
                ch = lineArray[i];
                ch = (char)('a' + (ch -'a'+ nKey) % 26);
                line[i] = ch;
            }
            out << lineArray;
        }
    }
    in.close();
    out.close();
}
```

Code Refactoring

✗ Use intention-revealing names! (Methods)

```
//This Functions opens text file , encodes the text and writes it to new file
static void bestFunctionEver(string fn){
    string lineArray;
    ifstream in(fn.c_str());
    ofstream out(("encoded"+fn).c_str());
    if (in)
    {
        while (getline( in, lineArray ))
        {
            for (int i=0;i<lineArray.length();i++){
                 if(lineArray[i]<='Z' && lineArray[i]>='A')
                    lineArray[i] = lineArray[i]-('Z'-'z');
             }

            int offset = 6;
            int nKey = offset%26;
            char ch;
            for (int i = 0; i < lineArray.length(); i++) {
                ch = lineArray[i];
                ch = (char)('a' + (ch -'a'+ nKey) % 26);
                line[i] = ch;
            }
            out << lineArray;
        }
    }
    in.close();
    out.close();
}
```

Code Refactoring

✓ Use intention-revealing names! (Methods)

```
static void openEncodeAndWriteToFile(string fn){
    string lineArray;
    ifstream in(fn.c_str());
    ofstream out(("encoded"+fn).c_str());
    if (in)
    {
        while (getline( in, lineArray ))
        {
            for (int i=0;i<lineArray.length();i++){
                 if(lineArray[i]<='Z' && lineArray[i]>='A')
                    lineArray[i] = lineArray[i]-('Z'-'z');
             }

            int offset = 6;
            int nKey = offset%26;
            char ch;
            for (int i = 0; i < lineArray.length(); i++) {
                ch = lineArray[i];
                ch = (char)('a' + (ch -'a'+ nKey) % 26);
                line[i] = ch;
            }
            out << lineArray;
        }
    }
    in.close();
    out.close();
}
```

Code Refactoring

✗ Use intention-revealing names! (Variables) (1)

```
static void openEncodeAndWriteToFile(string fn){
    string lineArray;
    ifstream in(fn.c_str());
    ofstream out(("encoded"+fn).c_str());
    if (in)
    {
        while (getline( in, lineArray ))
        {
            for (int i=0;i<lineArray.length();i++){
                 if(lineArray[i]<='Z' && lineArray[i]>='A')
                    lineArray[i] = lineArray[i]-('Z'-'z');
             }

            int offset = 6;
            int nKey = offset%26;
            char ch;
            for (int i = 0; i < lineArray.length(); i++) {
                ch = lineArray[i];
                ch = (char)('a' + (ch -'a'+ nKey) % 26);
                lineArray[i] = ch;
            }
            out<< lineArray;
        }
    }
    in.close();
    out.close();
}
```

Code Refactoring

✓ Use intention-revealing names! (Variables) (1)

```
static void openEncodeAndWriteToFile(string fileName){
    string lineArray;
    ifstream inputFile(fileName.c_str());
    ofstream outputFile(("encoded"+fileName).c_str());
    if (inputFile)
    {
        while (getline( inputFile, lineArray ))
        {
            for (int i=0;i<lineArray.length();i++){
                if(lineArray[i]<='Z' && lineArray[i]>='A')
                    lineArray[i] = lineArray[i]-('Z'-'z');
            }

            int offset = 6;
            int normalizedKey = offset%26;
            char curChar;
            for (int i = 0; i < lineArray.length(); i++) {
                curChar = lineArray[i];
                curChar = (char)('a' + (curChar -'a'+ normalizedKey) % 26);
                lineArray[i] = curChar;
            }
            outputFile << lineArray;
        }
    }
    inputFile.close();
    outputFile.close();
}
```

Code Refactoring

✗ Use intention-revealing names! (Variables) (2)

```
static void openEncodeAndWriteToFile(string fileName){
    string lineArray;
    ifstream inputFile(fileName.c_str());
    ofstream outputFile(("encoded"+fileName).c_str());
    if (inputFile)
    {
        while (getline( inputFile, lineArray ))
        {
            for (int i=0;i<lineArray.length();i++){
                 if(lineArray[i]<='Z' && lineArray[i]>='A')
                    lineArray[i] = lineArray[i]-('Z'-'z');
             }

            int offset = 6;
            int normalizedKey = offset%26;
            char curChar;
            for (int i = 0; i < lineArray.length(); i++) {
                curChar = lineArray[i];
                curChar = (char)('a' + (curChar -'a'+ normalizedKey) % 26);
                lineArray[i] = curChar;
            }
            outputFile << lineArray;
        }
    }
    inputFile.close();
    outputFile.close();
}
```

Code Refactoring

✓ Use intention-revealing names! (Variables) (2)

```
static void openEncodeAndWriteToFile(string fileName){
    string lineArray;
    ifstream inputFile(fileName.c_str());
    ofstream outputFile(("encoded"+fileName).c_str());
    if (inputFile)
    {
        while (getline( inputFile, lineArray ))
        {
            for (int i=0;i<lineArray.length();i++){
                if(lineArray[i]<='Z' && lineArray[i]>='A')
                    lineArray[i] = lineArray[i]-('Z'-'z');
            }

            int offset = 6;
            int normalizedKey = offset%26;
            char curChar, encodedChar;
            for (int i = 0; i < lineArray.length(); i++) {
                curChar = lineArray[i];
                encodedChar = (char)('a' + (curChar -'a'+ normalizedKey) % 26);
                lineArray[i] = encodedChar;
            }
            outputFile << lineArray;
        }
    }
    inputFile.close();
    outputFile.close();
}
```

Code Refactoring

✗ Function should be short

```
static void openEncodeAndWriteToFile(string fileName){
    string line;
    ifstream infile(fileName.c_str());
    ofstream outfile(("encoded"+fileName).c_str());
    if (infile)
    {
        while (getline( infile , line ))
        {
            for (int i=0;i<line.length();i++){
                if(line[i]<='Z' && line[i]>='A')
                   line[i] = line[i]-('Z'-'z');
            }

            int offset = 6;
            int normalizedKey = offset%26;
            char curChar, encodedChar;
            for (int i = 0; i < line.length(); i++) {
                curChar = line[i];
                encodedChar = (char)('a' + (curChar -'a'+ normalizedKey) % 26);
                line[i] = ch;
            }
            outfile << line;
        }
    }
    infile.close();
    outfile.close();
}
```

Code Refactoring

✓ Function should be short

```
static void openEncodeAndWriteToFile(string fileName){
    ifstream infile(fileName.c_str());
    ofstream outfile(("encoded"+fileName).c_str());
    if (infile)
    {
        encodeAndWriteLineByLine(infile, outfile);
    }
    infile.close();
    outfile.close();
}
static void encodeAndWriteLineByLine(ifstream& infile, ofstream& outfile){
    string line;
    while (getline( infile, line )){
        for (int i=0;i<line.length();i++){
            if(line[i]<='Z' && line[i]>='A')
                line[i] = line[i]-('Z'-'z');
        }
        int offset = 6;
        int normalizedKey = offset%26;
        char curChar, encodedChar;
        for (int i = 0; i < line.length(); i++) {
            curChar = line[i];
            encodedChar = (char)('a' + (curChar -'a'+ normalizedKey) % 26);
            line[i] = ch;
        }
        outfile << line;
    }
}
```

Code Refactoring

✗ Single Responsibility Principle!

```
static void openEncodeAndWriteToFile(string fileName){
    ifstream infile(fileName.c_str());
    ofstream outfile(("encoded"+fileName).c_str());
    if (infile)
    {
        encodeAndWriteLineByLine(infile, outfile);
    }
    infile.close();
    outfile.close();
}
static void encodeAndWriteLineByLine(ifstream& infile, ofstream& outfile){
    string line;
    while (getline( infile, line )){
        for (int i=0;i<line.length();i++){
            if(line[i]<='Z' && line[i]>='A')
                    line[i] = line[i]-('Z'-'z');
        }
        int offset = 6;
        int normalizedKey = offset%26;
        char curChar, encodedChar;
        for (int i = 0; i < line.length(); i++) {
            curChar = line[i];
            encodedChar = (char)('a' + (curChar -'a'+ normalizedKey) % 26);
            line[i] = ch;
        }
        outfile << line;
    }
}
```

Code Refactoring

✓ Single Responsibility Principle!

```
static void openEncodeAndWriteToFile(string fileName){
    string line;
    ifstream infile(fileName.c_str());
    ofstream outfile(("encoded"+fileName).c_str());
    if (infile)
    {
        encodeAndWriteLineByLine(infile, outfile);
    }
    infile.close();
    outfile.close();
}
static void encodeAndWriteLineByLine(ifstream& infile, ofstream& outfile){
    string line;
    while (getline( infile, line )){
        line = toLowerCase(line);
        line = encodeLine(line);
        outfile << line;
    }
}
static string toLowerCase(string line){
    for (int i=0;i<line.length();i++){
        if(line[i]<='Z' && line[i]>='A')
            line[i] = line[i]-('Z'-'z');
     }
     return line;
}
static string encodeLine(string line){...}
```

Code Refactoring

★ Final result (if final really exists)

```
static void openEncodeAndWriteToFile(string fileName){
    string line;
    ifstream infile(fileName.c_str());
    ofstream outfile(("encoded"+fileName).c_str());
    if (infile)
    {
        encodeAndWriteLineByLine(infile, outfile);
    }
    infile.close();
    outfile.close();
}
static void encodeAndWriteLineByLine(ifstream& infile, ofstream& outfile){
    string line;
    while (getline( infile, line )){
        line = toLowerCase(line);
        line = encodeLine(line);
        outfile << line;
    }
}
static string toLowerCase(string line){
    for (int i=0;i<line.length();i++){
         if(line[i]<='Z' && line[i]>='A')
            line[i] = line[i]-('Z'-'z');
     }
     return line;
}
static string encodeLine(string line){...}
```

Meaningful Names

- Intention-revealing names
- Good name length
- One notation style
- One word per one concept
- Meaningful names in domain context
- Meaningful names in self context

Meaningful Names

- Intention-revealing names
- Good name length
- One notation style
- One word per one concept
- Meaningful names in domain context
- Meaningful names in self context

```
int d; // elapsed time in days
int ds;
int dsm;
int faid;
```

⇓

```
int elapsedTimeInDays;
int daysSinceCreation;
int daysSinceModification;
int fileAgeInDays;
```

Meaningful Names

- Intention-revealing names
- Good name length
- One notation style
- One word per one concept
- Meaningful names in domain context
- Meaningful names in self context

```
type CustomersListWithAllCustomers;
type list;
```

$\Downarrow$

```
type allCustomers;
type customersInOrder;
```

Meaningful Names

- Intention-revealing names
- Good name length
- One notation style
- One word per one concept
- Meaningful names in domain context
- Meaningful names in self context

```
const int maxcount = 1;
bool change = true;
class Repository;
private: string NAME;
class personaddress;
void getallorders();
```

$\Downarrow$

```
const int MAXCOUNT = 1; //Constants
bool isChanged = true; //Booleans
class IRepository; //Interface
private: string _name;//Private member
class PersonAddress; //Class
void getAllOrders(); //Method
```

Meaningful Names

- Intention-revealing names
- Good name length
- One notation style
- One word per one concept
- Meaningful names in domain context
- Meaningful names in self context

```
//1.
Data loadSingleData();
Data fetchDataFiltered();
Data getAllData();
//2.
void setDataToView();
void setObjectValue(int value)
```

⇓

```
//1. Concept get()
Data getSingleData();
Data getDataFiltered();
Data getAllData();
//2. Concept load()
void loadDataToView();
//3. Concept set(value)
void setObjectValue(int value);
```

Meaningful Names

- Intention-revealing names
- Good name length
- One notation style
- One word per one concept
- Meaningful names in domain context
- Meaningful names in self context

```
class EntitiesRelation{
    Entity o1;
    Entity o2;
}
```

⇓

```
class StoreItem{
    Entity product;
    Entity category;
}
```

Meaningful Names

- Intention-revealing names
- Good name length
- One notation style
- One word per one concept
- Meaningful names in domain context
- Meaningful names in self context

```
string addressCity;
string addressHomeNumber;
string addressPostCode;
```

⇓

```
class Address{
    string _city;
    string _homeNumber;
    string _postCode;
}
```

Some guidelines for methods

- Small!
  - They should even be smaller!
  - Indent level not higher than two nested structures
  - One liner in if/else/while/for
  - Coarse guideline: $\leq$ 15 lines
- SRP
  - Single Responsibility Principle
  - Do one thing and one thing only
  - Do that thing well
- Stepdown rule
  - Function is followed by functions of next abstraction level
  - Code can be read from top to bottom

Some guidelines for methods

- Small!
  - They should even be smaller!
  - Indent level not higher than two nested structures
  - One liner in if/else/while/for
  - Coarse guideline: $\leq$ 15 lines
- SRP
  - Single Responsibility Principle
  - Do one thing and one thing only
  - Do that thing well
- Stepdown rule
  - Function is followed by functions of next abstraction level
  - Code can be read from top to bottom

Some guidelines for methods

- Small!
  - They should even be smaller!
  - Indent level not higher than two nested structures
  - One liner in if/else/while/for
  - Coarse guideline: $\leq$ 15 lines
- SRP
  - Single Responsibility Principle
  - Do one thing and one thing only
  - Do that thing well
- Stepdown rule
  - Function is followed by functions of next abstraction level
  - Code can be read from top to bottom

Some guidelines for methods

- Small!
  - They should even be smaller!
  - Indent level not higher than two nested structures
  - One liner in if/else/while/for
  - Coarse guideline: $\leq$ 15 lines
- SRP
  - Single Responsibility Principle
  - Do one thing and one thing only
  - Do that thing well
- Stepdown rule
  - Function is followed by functions of next abstraction level
  - Code can be read from top to bottom

Some more principles

- Boy's Scout Rule: Leave the place better than you found it
- KISS: **K**eep **I**t **S**imple **S**tupid
- YAGNI: **Y**ou **A**re **N**ot **G**onna **N**eed **I**t
- DRY: **D**on't **R**epeat **Y**ourself
- SOC: **S**eperation **O**f **C**oncerns

Contents

1. Introduction

2. Clean Code
   2.1 Code Refactoring Example
   2.2 Naming
   2.3 Methods
   2.4 General principles

3. Famous software bugs

4. References

Bugs which could have been avoided through clean coding

- 2002 study commissioned losses in the US economy through bugs $\approx$ 59.5 billion/year
- `http://royal.pingdom.com/2009/03/19/10-historical-software-bugs-with-extreme-consequences/`
- Ariane 5 `http://www.ima.umn.edu/~arnold/disasters/ariane.html`
- Patriot Missile Failure `http://www.ima.umn.edu/~arnold/disasters/patriot.html`
- Loss of Mars Climate Orbiter `http://marsprogram.jpl.nasa.gov/msp98/news/mco990923.html`
- `http://thedailywtf.com/`

Contents

1. Introduction

2. Clean Code
   - 2.1 Code Refactoring Example
   - 2.2 Naming
   - 2.3 Methods
   - 2.4 General principles

3. Famous software bugs

4. References

Links and References

- **Martin, Robert C.**, *Clean Code: A Handbook of Agile Software Craftsmanship*
- Slide 5: `http://www.osnews.com/story/19266/WTFs_m`