Self Expressive Code

A handbook of write readable codes

```
if (dir == FORWARD) {
    through();
}
if (dir == TURN_LEFT) {
    turn(RIGHT);
    turn(RIGHT);
    turn(RIGHT);
}
```

Stephen Wang

Self-Expressive Code

A handbook of write readable code.

Stephen Wang

This book is for sale at http://leanpub.com/self_expressive_code

This version was published on 2014-04-06

Tweet This Book!

Please help Stephen Wang by spreading the word about this book on Twitter!

The suggested hashtag for this book is #Software Programming, Code Readability.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#Software Programming, Code Readability

Contents

Introduction

When we are writing programming code, readability issues happen. People often face some hard to read source codes. What kind of codes are lower readability? What caused this happen? How to solve these issues? All the above questions are big.

Restricted by the programming language, codes can be written in predefined ways. People might think it is not possible to write source code as certain language. That's true. We could not write certain language, then let computer analyze the commands and run properly, but we can write source code as certain language as we can. Programming language provides some useful symbols allow programmers to write source code easy to understand.

Some issues caused by English, some non-native speakers have smaller vocabulary set, they tend to use simple words only to express what they want to do. However, even for native speakers, they can give good names to classes, methods or variables, but when call these classes, methods, variables, problems also appear. That means, code readability is no only depend on the names, but also depend on how it will be called. Plus, the algorithm, the structure, the architecture will affect the readability of codes.

Improve code's readability is not so easy.

In this book, I would like to introduce a new way to write source code - 'Self-Expressive Code'. As implied in its name, Self-Expressive Code means, the code itself can express what it will do. Programmers can understand the meaning of source code faster and easier. Self-Expressive Code is chasing for the way to write source code that could be understood at a glance.

Java is still popular and influential. This book picked Java as its sample language.

Acknowledgement

There is a list who helped editing this book.

Michael Chen Liu Chengzhang Han Kuikui

Thank my family, without their support, this book could not be published.

The first edition of this book is published in Chinese on paper. The initial intention for this book is for help non-English speakers write more readable codes. Michael has reviewed the first edition and provided some useful suggestions. He also provided many useful ideas for the second edition. This is the second edition, chapters are re-organized, samples are revised. I hope this book can benefit programmers across the world, not only non-English speakers.

This book referred the following books:

- Design Patterns:Elements of Reusable Object-Oriented Software[1], Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides ISBN: 9787111075752
- Refactoring[2], Martin Folwer
- Clean Code[3], Robert Martin
- The Art of Readable Code (Theory in Practice)[4], Dustin Boswell, Trevor Foucher

[1] http://en.wikipedia.org/wiki/Design_Patterns

[2] http://martinfowler.com/books/refactoring.html

[3] CleanCode:AHandbookofAgileSoftwareCraftsmanship

[4] http://baike.baidu.com/view/8805043.htm

What Is Self-Expressive Code

In assembly language, codes are written for machines, people can hard to read source code and guess its meaning. After decades, programming languages become more and more easy to read. Shortly, codes are written for people.

However, in the real world, we could hard to read many codes in C# or Java, even certain words are used to name objects in programming. In a opened source famous platform, one module contains a bunch of lower readable codes. After read that code, it seems the authors of the codes tend to write source code for machines, not for people. We cannot know the real reason, but we could analyze the source code.

The source code has:

1. if-else conditions that do not evaluate same object.
2. non-English names.
3. long branches/methods.
4. many duplications/similarities.

Those above features show that the programmer were not pay enough attention about the reader's feeling. This third party company shared their source code on that famous platform for brand image or something. Unfortunately, the source code just leads to the opposite way. After read that code I started to worry about the quality of their health care products. Hope they have different process to make the products quality higher.

So, what is good way to write readable source code?

For instance, in a game application, player can control a hero to fight the enemy in a scene. The code could seem like:

```
scene.fight(hero, enemy, Kongfu.FIST);
```

In the code above, readers cannot understand the fight direction between hero and enemy.

An revised version could seem like:

```
hero.fight(enemy, Kongfu.FIST);
```

For the restriction of programming language, constant should have its class name. The meaning of this code is clear, but the code still seems like to be a machine language, not human language. Then the ace version is appear:

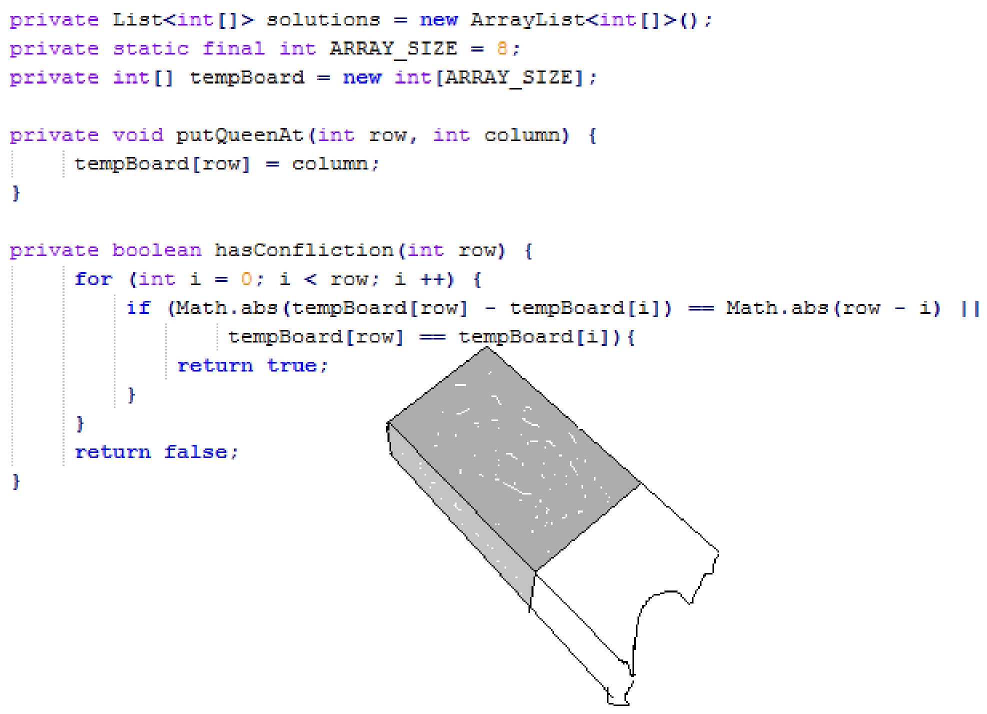
```
hero.fight(enemy).with("FIST");
```

The fight() method returns a Fight object that has with() method. The with() method passed by a Kongfu name that allows a KongfuFactory to create a Kongfu object and calculate the damage. This process makes more classes, but it also make the source code more readable.

Implement functionality is the basic responsibility of source code, but keep it readable, extensible, changeable are also important. As mentioned in the head of this section, codes are written for people. I suggest to write source code as certain language as possible. I named that method 'Self-Expressive Code'. Before the word 'Self-Expressive Code' appear in the world, there are two similar words: Self-Documenting Code[5] and Self-Describing Code[6] already existed. They might have the same meaning, but I think the word 'expressive' is more expressive, so I named this way of writing code as 'Self-Expressive Code'.

[5]http://en.wikipedia.org/wiki/Self-documenting
[6]http://en.wikipedia.org/wiki/Self-documenting

Part I Readability



```java
private List<int[]> solutions = new ArrayList<int[]>();
private static final int ARRAY_SIZE = 8;
private int[] tempBoard = new int[ARRAY_SIZE];

private void putQueenAt(int row, int column) {
    tempBoard[row] = column;
}

private boolean hasConfliction(int row) {
    for (int i = 0; i < row; i ++) {
        if (Math.abs(tempBoard[row] - tempBoard[i]) == Math.abs(row - i) ||
                tempBoard[row] == tempBoard[i]){
            return true;
        }
    }
    return false;
}
```

Chapter 1 Annoyance of Lower Readability

When programmers read others' code, they would have feelings about the code. Just like users access a website. If website makes user unsatisfied, user will go away. If the code is not good for read, what would the reader do? The most frequently occurred thing might be spitting out dirty words. The worst thing is that after spit out, the reader noticed that the source were written by himself/herself. Apparently, lower readable codes make programmers unsatisfied.

Lower readable codes may be a:

1. **Mouse Wheel Tester**

   Long file, long method, long branch, those force readers to roll their mouse wheel continuously. The source code, in other side, is a mouse wheel tester.
2. **Eye Tester**

   Similar codes make user confused about the nuances. Similar codes force user to look inside with caution.
3. **Memory Tester**

   Name variables as a, b, c or count*1, count*2 to force readers remember meaning of these variables.
4. **Patience Tester**

   Long names hide the key information that forces user stop and think. Unformatted source make user to keep big patience to read. Deep nested source makes reader think the relationship harder.

or, it may have such features:

1. **Names Make No Sense**

   Method names like : doProcess, invokeMethod, executeService.

   Variable names like : returnValue, tempValue, list, result.
2. **Dialects**

   Using other languages other than English, makes other readers totally have no idea about the names. Even they share the same language.

   'YouYu' in Chinese have at least eight possible meanings.

   'ikou' in Japanese have at least four possible meanings.
3. **Made Up Words**

   Using ambiguous words that misguide readers, such as 'float DBL'. That leads readers to think 'DBL' is the abbreviation of 'double'.
4. **Quiz**

   Using complicated expressions to make things more complex.

Lower readability also makes code hard to change, hard to extend, hard to fix bugs. Lower readability codes share one feature - forces readers stop and think, that causes lower productivity, lower quality.

1.1 Annoying Bugs

Programmers are inevitable to face bugs. If it is easy to be fixed, programmers might not hate bugs too much. In the real world, however, bugs are hard to be reproduced, hard to be traced, hard to be fixed, or easy to degrade.

Not reproducible

One possible reason is that there are too many factors to make it happens again. And these factors are existing in the source code, but not in the business process. This reflects the code appended inner factors, and if possible, eliminate them to make bugs reproducible.

Not traceable

One possible reason is that the bug caused by one complicated line code. If user got an 'NullPointerException' at the following code. Which is the root cause of the problem?

```
product.setUpdater(user.getFullName());
```

Another possible reason is that the bug were produced by mis-use of third party framework. For example, Spring framework allows user to configure their xml to run system, but it is not easy to find if xml has mistake.

Third possible reason is that the error message does not indicate the real reason. For example, Ebean allows user to access db easier. If the server name of Ebean is not set properly, User would get a message of 'NullPointerException'. and user could not easily to solve this problem without reading the source code of Ebean.

Not Fixable

After tried so hard, programmer found the reason of the source code, but cannot fix that bug. Because it is a very influential place. If it changes, too much tests are required, but time is limited.

Degrade

After fixed the bug, it works. Unfortunately, another bug appears according to this fix.

All those above annoyance occur frequently with lower readable codes.

1.2 Hard To Understand

When a new programmer joined to a team, he/she needs to read source code before touch it. If the source code has lower readability, it takes long time to understand. Until the new programmer understood the source code, he/she could do nothing about fix bugs or apply changes. This also works for job transfer from a member to another. It depends on the readability, if the codes are easy to read, the time be taken is shorter, otherwise, longer time needed.

There is a list that frequently happened in lower readable codes.

Negatives

Negative words contain a 'NOT' information, if put negative variable into a conditional statement, it just like a quiz : **not of not is positive**. If it happens to be a non-adjective variable, the code becomes harder to read.

```
if ((hardKeyboardHidden == 1) == false)
```

Circular Reference

If object A contains an object B, and object B for some reasons needs reference from object A, the two objects have a circular reference, just like below:

```
class Container {
    Context context;
    Product product;
}
```

```
class Proudct {
    //This is for calling of 'container.context'
    Container container;
}
```

In this case, **container.product.container.product** would be a strange object. In fact this is caused by wrong ownership.

Diverse Meanings

If a local variable in a method have diverse meanings, readers could not easily check the meaning of the variable at a line. The variable 'flag' in the following code is hard to understand.

```java
public String registerLesson(long userId, long lessonId) {
    int flag = 0;

    flag = checkUserExist(userId);

    if (flag == 0) {
        return "User does not exist.";
    }

    flag = checkLessonExist(lessonId);

    if (flag == 0) {
        return "Lesson does not exist.";
    }

    return "";
}
```

Diverse Branches

> Branches may have conditions, but the conditions should have same meaning, different values. If two connected branches have different conditions, it makes no sense.

```java
public void method(int flag, String name) {
    if (flag == 0) {
        //process 1;
    } else if (name.equals("abc")) {
        //process 2;
    }
}
```

Ambiguous Names

> If the name of a method can not show its meaning, reader should read every and each line of its implementation to determine its real functionality. That takes time.
>
> The following is a sample of ambiguous name.

```java
public class Keyboard {
    //Pressed the 'alt' key or alter to another keyboard layout?
    public void alt() {
    }
}
```

Non-Match Documents

Many maintenance engineers are confusing about that the source code have nor creditable documents, neither comments, which means the code is the only document. Unfortunately, most of these time, the source code is not readable too.

1.3 Hard To Change Or Extend

Software changes. If a requirement change comes, how much time will be taken? The higher the changeability is, the shorter the time is needed.

Changeability

A widget[7] place module in a system have magnet effects for widgets. That algorithm is changed frequently because widget have many sizes. But the code is hard to be changed, because the 'move' method is too complicated:

```java
private Widget[] widgets;

private int focus;

public void move(int dir) {
    if (dir == 1) {// move right
        if (widgets[focus].width == 1) {
            int next = 0;
            next = findNext();
            if (widgets[next].height == 2) {
                if (widgets[next].y == widgets[focus].y) {
                    //omitted
                }
                //..omitted.
            } else if (widgets[next].height == 3) {
                if (widgets[next].y == widgets[focus].y) {
```

[7]http://developer.android.com/guide/topics/appwidgets/index.html

```
                        //omitted
                    }
                    //..omitted.
                }
            }
    } else (dir == -1) {  //move left
        //omitted
    }
}
```

In this complicated algorithm, programmer try to enumerate every possibility about moving widget. Any new possibility will extend the method. The final version of source is very terrible, and messed up. Even a small change to this algorithm will cause big modification.

Extensibility

An input method system on android platform is required to have new input device : remote control and game pad.

Below is the skeleton of the implemented codes:

```
public class Keyboard {
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (hardKeyboardConnected == 0) {
            //treat with soft keyboard
        } else {
            //treat with hard keyboard
            if (keyCode >= KeyEvent.KEYCODE_A
                && keyCode <= KeyEvent.KEYCODE_Z) {
            } else if (keyCode == KeyEvent.KEYCODE_ENTER) {
            } else if (keyCode == KeyEvent.KEYCODE_SPACE) {
            } else if (keyCode == KeyEvent.KEYCODE_SHIFT) {
            }
        }
    }
}
```

When add device like remote control and game pad, because it is not detectable for android system, like a physical keyboard plugged in.

The code becomes:

```java
public class Keyboard {
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (hardKeyboardConnected == 0) {
            //treat with soft keyboard
        } else {
            //treat with hard keyboard
            if (keyCode >= KeyEvent.KEYCODE_A
                && keyCode <= KeyEvent.KEYCODE_Z) {
            } else if (keyCode == KeyEvent.KEYCODE_ENTER) {
            } else if (keyCode == KeyEvent.KEYCODE_SPACE) {
            } else if (keyCode == KeyEvent.KEYCODE_SHIFT) {
            } else if (keyCode >= KeyEvent.KEYCODE_BUTTON_1
                && keyCode <= KeyEvent.KEYCODE_BUTTON_10) {  //for remote control
            } else if (keyCode >= KeyEvent.KEYCODE_BUTTON_A
                && keyCode <= KeyEvent.KEYCODE_BUTTON_C) {  //for game pad
            } else if (keyCode >= KeyEvent.KEYCODE_BUTTON_X
                && keyCode <= KeyEvent.KEYCODE_BUTTON_Z) {  //for game pad
            }
        }
    }
}
```

If follow this trend to modify source code, the source code will be very fatty and buggy.

All of the above system has lower readability.

1.4 Easy To Make Bugs In.

Not Match

It may cause bug when call a one method that declared one behavior, but actually did two.

```java
public void update() {
    reloadData();
    saveFile();
}
```

Duplications/Similarities

It may cause bug in case that should call method A, but called a similar method B.

```java
public void clear() {
    //remove all elements
}
```

```java
public void clean() {
    //remove all dump data.
}
```

Ambiguous

It is confused when trying to call a method with ambiguous name and implementation.

```java
public int[] countDifferentSeries(int[][] basketTwoDArray) {
    int count = 0;
        //omitted.
    return differentSeriesCount;
}
```

This method start with ‘count’, but returns an array. It is not easy to understand the meaning of this code in short. If want to count each item in the array, how about to calculate separately in a loop?

Complicated

Deeper nested codes makes code hard to read, hard to understand.

```java
private Widget[] widgets;

private int focus;

public void move(int dir) {
    if (dir == 1) {// move right
        if (widgets[focus].width == 1) {
            int next = 0;
            next = findNext();
            if (widgets[next].height == 2) {
                if (widgets[next].y == widgets[focus].y) {
                    //omitted
                }
                //..omitted.
            } else if (widgets[next].height == 3) {
                if (widgets[next].y == widgets[focus].y) {
                    //omitted
                }
                //..omitted.
            }
        }
    } else (dir == -1) {  //move left
        //omitted
    }
}
```

When touch this code, it is very easy to make bugs in.

1.5 Hard To Refactor

Inconsistent Style

Many programmers do not care about the style of source code. In their editor both of below exist.

```
a=a+1;
```

```
a = a + 1;
```

In this case, if reader want to search something, they do not know which pattern should they use. They have to add additional information into the search box - Make it regular expression.

Also, the mixed style exist together makes new joined programmers confused - What rule should I follow if I want to change the source code.

Inconsistent Words

Not only the coding style, if the terms or words are not inconsistent in one system, readers will also be confused.

In one file, ‘save’ means ‘insert’, but in another file, ‘save’ means ‘update’, so why not use ‘insert’ and ‘update’ to avoid this?

Inconsistent Process

Besides, in a system, treat similar objects in different ways may take extra time to understand the difference. In a input method system, typed alphabet will be transferred to certain characters according to the selected transfer rules.

The following code shows the design simply.

```java
public interface TransferEngine {
    public List<String> transfer();
}
```

```java
public class InputMethodService {
    public List<String> transfer() {
        String text = getTypedAlphabets();
        TransferEngine engine = getSelectedTransferEngine();
        List<String> candidates = engine.transfer(text);
        return candidates;
    }
}
```

If added one rule, it should be a implementation of TransferEngine,

```java
public class HiraganaEngine implements TransferEngine {
    public List<String> transfer(String text) {
        //...
    }
}
```

However, the author changed code as follows:

```java
public class InputMethodService {

    public List<String> transfer() {
        String text = getTypedAlphabets();
        TransferEngine engine = getSelectedTransferEngine();

        List<String> candidates = null□

        if (engine instanceof SymbolEngine) {
            candidates = new ArrayList<String>();
            candidates.add(text);
            return candidates;
        }

        candidates = engine.transfer(text);
        return candidates;
    }
}
```

Further Reading