Code Cleaning

some techniques for improving existing code

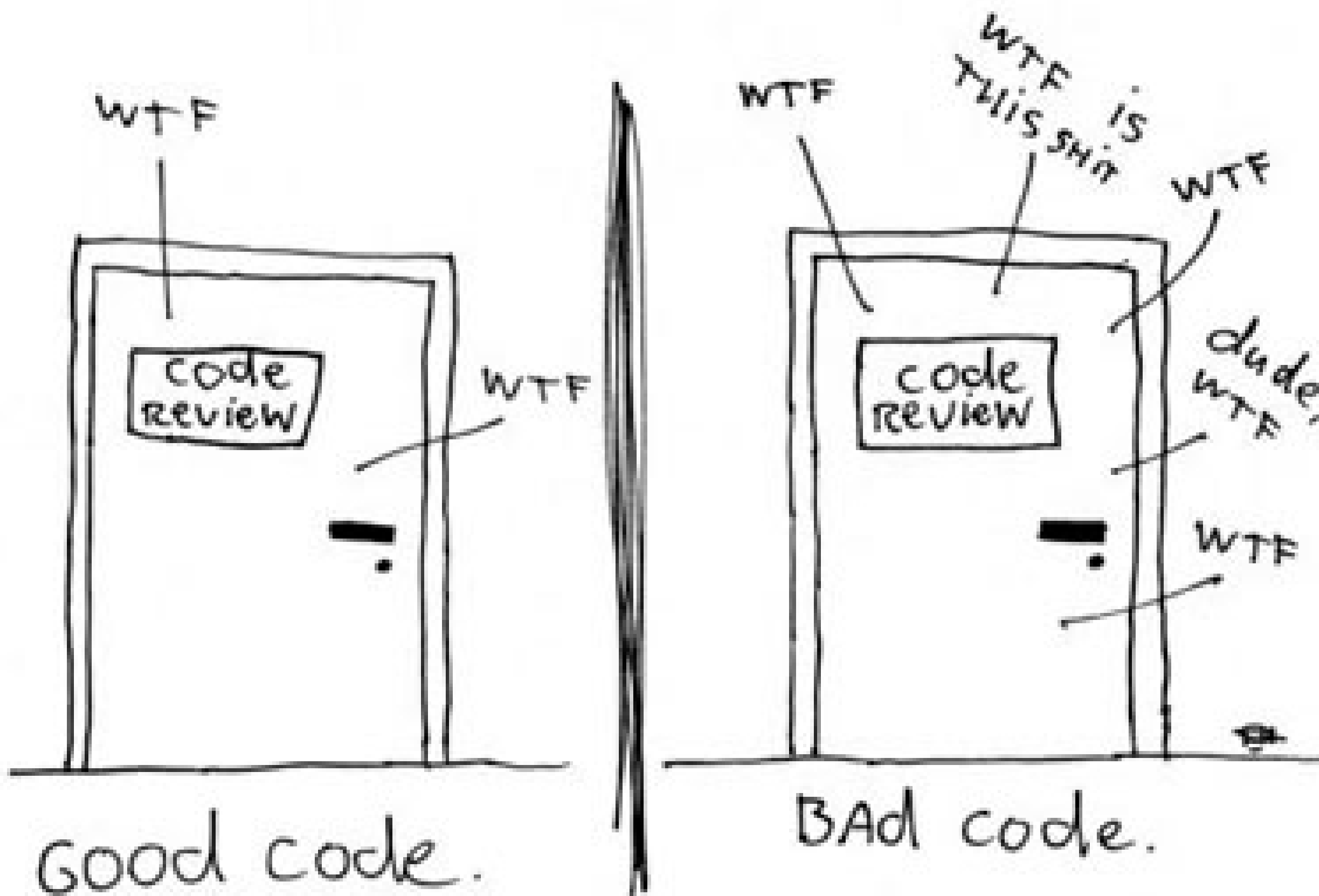
A fundamental condition for large scale agile and effective software development is that the codebase is in a healthy state and easy to work with. If you do not take care of your codebase it will rot and your project (and company) will probably fail.

Clean code is code that looks like it is written by somebody who cares, and where there is nothing obvious that you can do to make it better(*). This talk will discuss some techniques and tricks for code cleaning; it might give you an idea about how to improve existing code, how to keep your codebase healthy.

Olve Maudal
oma@pvv.org

10 minute lightening talk at Smidig 2008
October 9-10, 2008

(*) Michael Feathers' definition



(reproduced with kind permission of Thom Holwerda)

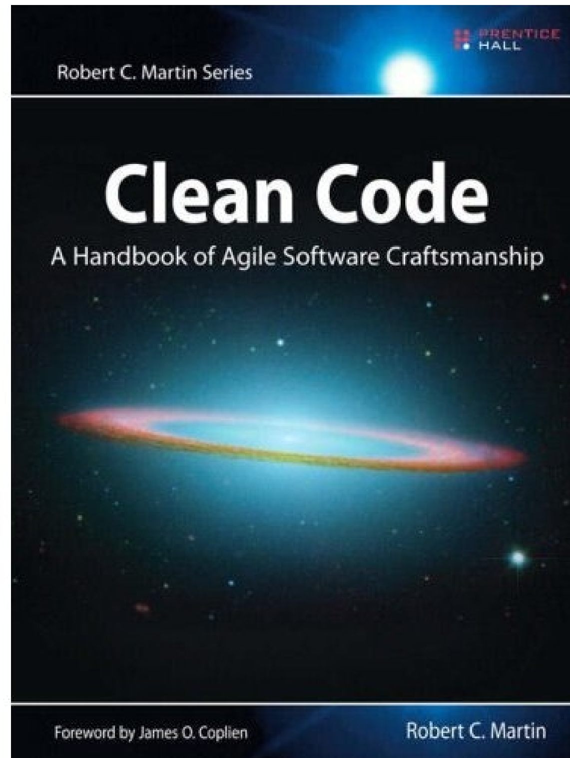
The Boy Scout Rule

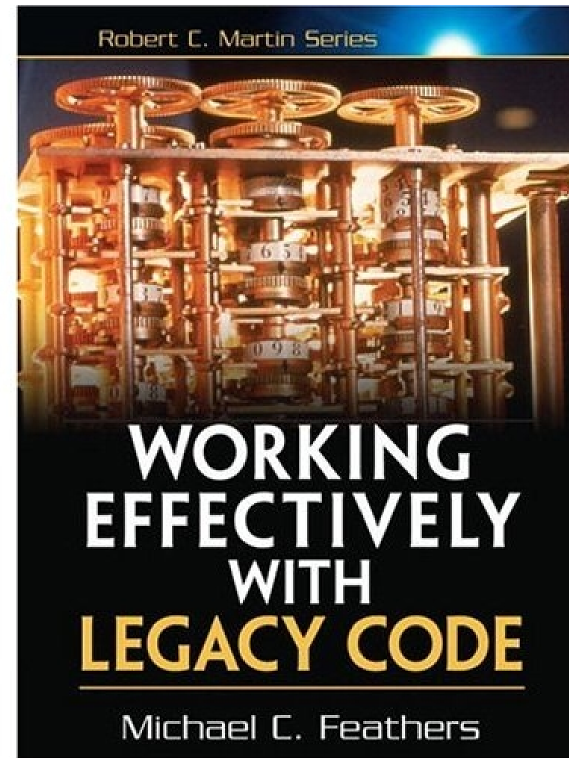
Leave the campground cleaner than you found it.

(see CCp14)

Background and Disclaimer

This talk is very much inspired by Uncle Bob's latest book about writing clean code, but also by Michael Feathers book about working with dirty code.





Many examples, sentences and ideas in this talk are just ripped out from these excellent books.

I agree with most of the stuff I present here...

Conditionals

Current solution:

```
if (!isInvalid(value)) {
   ...
}
```

Negatives are harder to understand than positives.

Possible improvement:

```
if (isValid(value)) {
   ...
}
```

Explanatory variables

Current solution:

```
boolean isLeapYear(int year) {
  return ((year % 4 == 0) && (year % 400 == 0)) ||
         ((year % 4 == 0) && (year % 100 != 0));
}
```

The code above does the right thing, but it is possible to improve the readability.

Possible improvement:

```
boolean isLeapYear(int year) {
  boolean fourth = year % 4 == 0;
  boolean hundreth = year % 100 == 0;
  boolean fourHundreth = year % 400 == 0;
  return fourth && (!hundreth || fourHundreth);
}
```

Functions

A function should ideally do just one thing, and do it well. Above is an example of a function that does many things. If a function does more than one thing, consider splitting it.

Current solution:

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

Possible improvement:

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

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

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

Try/Catch Blocks

Current solution:

```
public void delete(Page page) {
  try {
    deletePage(page);
    registry.deleteReferences(page.name);
    configKeys.deleteKey(page.name.makeKey());
  } catch (Exception e) {
    logger.log(e.getMessage());
  }
}
```

This code also try to do several things. Delete a page, exception handling and logging. This code might be improved by splitting up into three functions.

Possible improvement:

```
public void delete(Page page) {
  try {
    deletePageAndAllReferences(page);
  } catch (Exception e) {
    logError(e);
  }
}

private void deletePageAndAllReferences(Page page) throws Exception {
  deletePage(page);
  registry.deleteReferences(page.name);
  configKeys.deleteKey(page.name.makeKey());
}

private void logError(Exception e) {
  logger.log(e.getMessage());
}
```

Naming

There are two issues with this code.

The first issue is that the names are probably not at the right level of abstraction. It kind of assumes that this is a phone modem and that only a phone number is a valid way of dialing. You should consider either to change the name of the class to PhoneModem, or change the argument name for dial.

Current solution:

```
interface Modem
{
  public void dial(String phoneNumber);
  public void hangup();
  public void send(char c);
  public char recv();
}

class MyModem implements Modem {
  // ...
}
```

Possible improvement:

```
interface Modem
{
  public void dial(String connectionLocator);
  public void hangup();
  public void send(char c);
  public char recv();
}

class MyModem implements Modem {
  // ...
}
```

Responsibility

The other issue is that this class violates the Single Responsible Principle by having two responsibilities; connection management and data communication.

Current solution:

```
interface Modem
{
  public void dial(String connectionLocator);
  public void hangup();
  public void send(char c);
  public char recv();
}

class MyModem implements Modem {
  // ...
}
```

Possible improvement:

```
interface DataChannel {
  public void send(char c);
  public char recv();
}

interface Connection {
  public void dial(String connectionLocator);
  public void hangup();
}

class MyModem implements DataChannel, Connection {
  // ...
}
```

Commands and Queries

A function should either do something or answer something, not both. It is sometimes better to separate into a query and a command.

Current solution:

```
interface Iterator {
    Object next();
    bool isDone();
    // ...
}

// ...

while ( !iterator.isDone() ) {
    Object o = iterator.next();
    // ... do stuff with the object
}
```

Possible improvement:

```
interface Iterator {
    Object current();
    void advance();
    bool isDone();
    // ...
}

// ...

while ( !iterator.isDone() ) {
    Object o = iterator.current();
    // ... do stuff with the object
    iterator.advance();
}
```

Comments

Current solution:

```
// should be deleted?
if (timer.hasExpired() && !timer.isRecurrent()) {
  ... do stuff to delete timer ....
}
```

Code with lots of comments are usually a sign of poor craftmanship. Comments are lies. Uncommented code is usually much better than commented code. If you find a comment, try to get rid of it.

Possible improvement:

```
private boolean shouldBeDeleted(Timer timer) {
  return timer.hasExpired() && !timer.isRecurrent();
}

...

if (shouldBeDeleted(timer)) {
  ... do stuff to delete timer ...
}
```

The 7 tricks that we have discussed is just a few out of hundreds of similar small improvements that you should consider.

Writing clean code is important, but it is also important to clean up code as you see the opportunity.

Remember the boy scout rule?

The Boy Scout Rule

Leave the campground cleaner than you found it.

(see CCp14)

If you do not take care of your codebase it will rot, and your project (and maybe your company) will probably fail.

Analogy

The codebase is like a kitchen

suppose you are just going to make something nice for yourself

then, really, anything will do.

Even...

but, modern software development is usually about more than just making something nice for yourself.

It is usually about making something really fancy...

寿司

⁺⁺sushi⁺⁺

which one do you like best?

together with a large team...

for some demanding customer...

Then it is obvious:

To succeed you need a clean and functioning working environment.

OLIVE OIL
RHODODENDRON
SPIDER CRAB

Your codebase is like a kitchen.

Keep it clean so that you can create spectacular solutions for your demanding customers!

!

The Boy Scout Rule

Leave the campground cleaner than you found it.

(see CCp14)

The Importance of hygiene

Uncle Bob has suggested that we are now about to "discover" techniques and principles in software engineering that can be compared to the discovery of the importance of hygiene in hospitals by Ignaz Semmelweis in the middle of the 19th century.

(Semmelweis in the middle with arms crossed)

Semmelweis found that by introducing hand washing standards before surgery the number of fatal incidence caused by diseases dropped drastically. At the time, diseases were attributed to many different unrelated causes. Each case was considered unique, just like a human person is unique. Semmelweis' hypothesis, that there was only one cause, that all that mattered was cleanliness, was extreme at the time, and was largely ignored, rejected or ridiculed. [wikipedia]

At the point where a certain standard hygiene was accepted and enforced by the medical establishment, doctors started to behave as a group of professionals. This happened about 60 years after Semmelweis' discovery. As software engineers we are not always behaving like professionals, especially not at times where we let pressure from management and customers decide whether we write clean code or not. We know that dirty code is going to slow us down and delay the project, but still, for some reason, we sometimes end up in situations where we do exactly what we are not supposed to do. Imagine how a group of doctors today would react to a situation where they are told not to wash their hands between surgeries? Doctors act as professionals. Unfortunately, as a group of software engineers, we are not there... yet.