
Clean Code #1

cuong@techmaster.vn

Clean Code

Đối tượng và cấu trúc dữ liệu -
Object Data Structure

Kiểm thử - Unit Test

Code thối - Code Smell

Định dạng mã nguồn
- Code Format

Tham khảo

Quy tắc chung - General Rules

Quy tắc thiết kế - Design Rules

Quy tắc đặt tên - Naming Rules

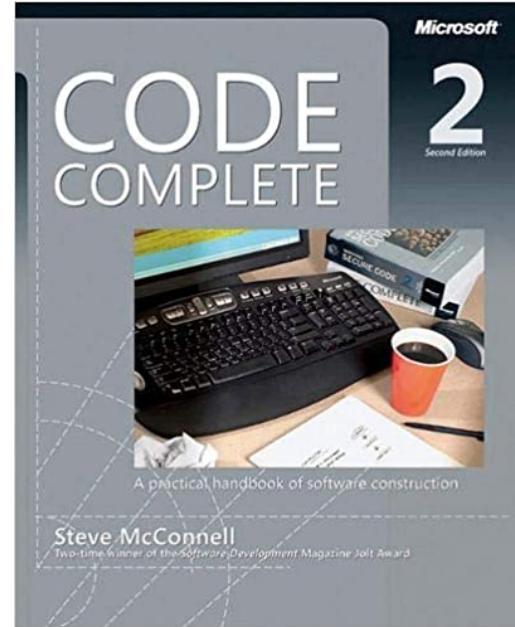
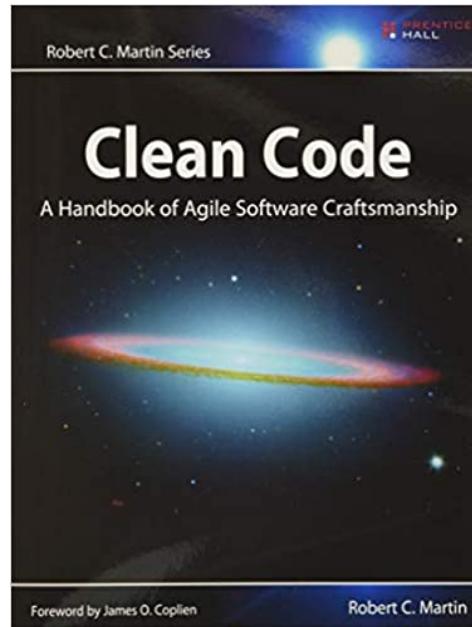
Hàm - Function

Chú thích - Comments

Code dễ hiểu - Unstandability Tips

Clean Code là gì?

Clean Code tên sách “Clean Code: A Handbook of Agile Software Craftsmanship” – 2008 của Robert C Martin. Trước đó có một quyển sách tương tự **Code Complete** – 2004 của Steve Mc Connell





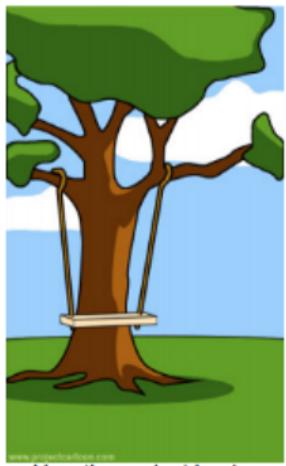
Robert Cecil Martin, nick Uncle Bob là một kỹ sư phần mềm kỳ cựu, giảng viên và tác giả vài tựa sách dạy lập trình ăn khách. Ông đúc kết những quy tắc để viết mã tốt hơn, dễ đọc, dễ bảo trì hơn. Ông còn là sáng lập viên tổ chức Agile Manifesto và chủ biên tạp chí “C++ Report”, chủ tịch Agile Alliance.

5 December 1952

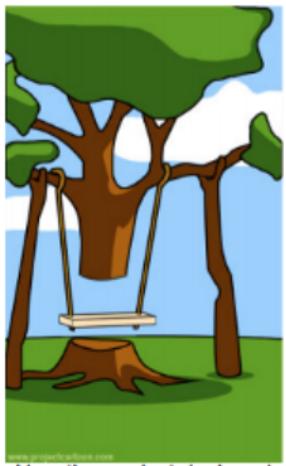
Clean Code + Agile



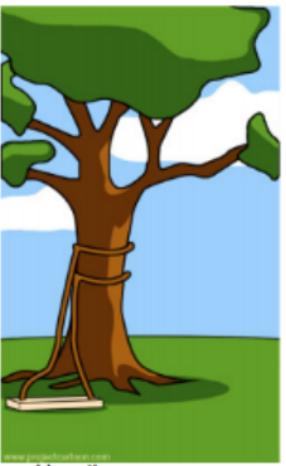
How the customer explained it



How the project leader understood it



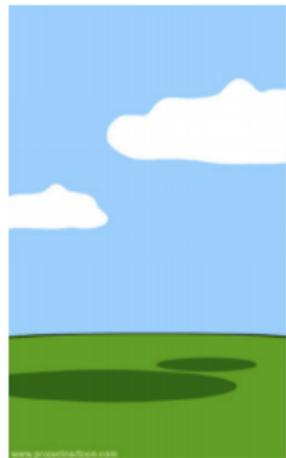
How the analyst designed it



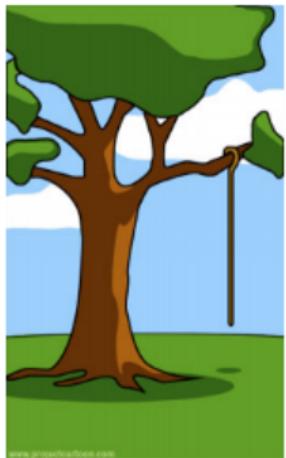
How the programmer wrote it



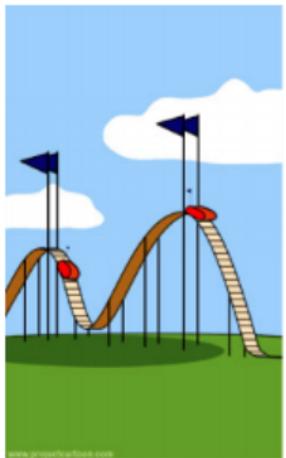
How the business consultant described it



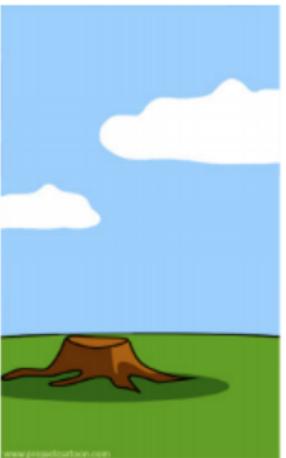
How the project was documented



What operations installed



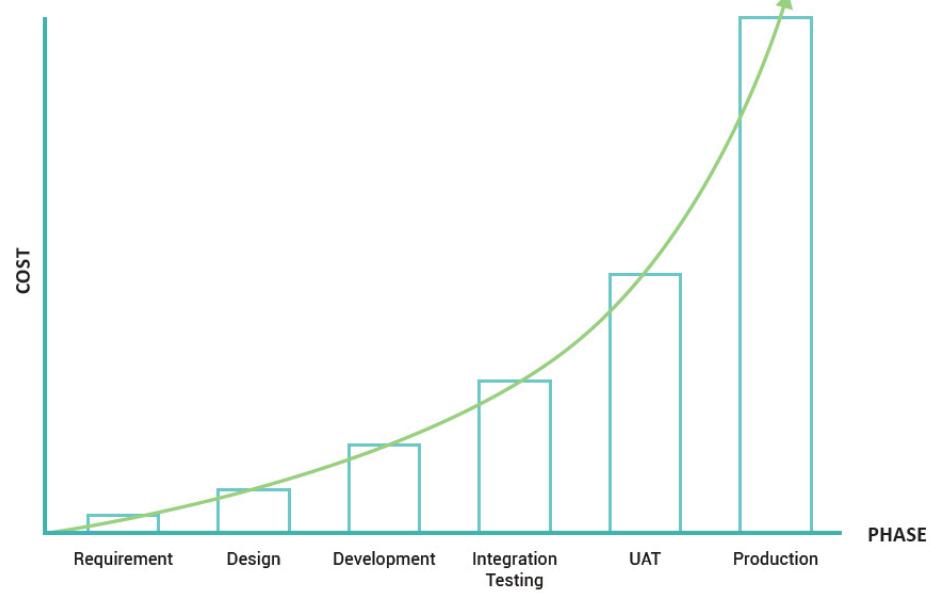
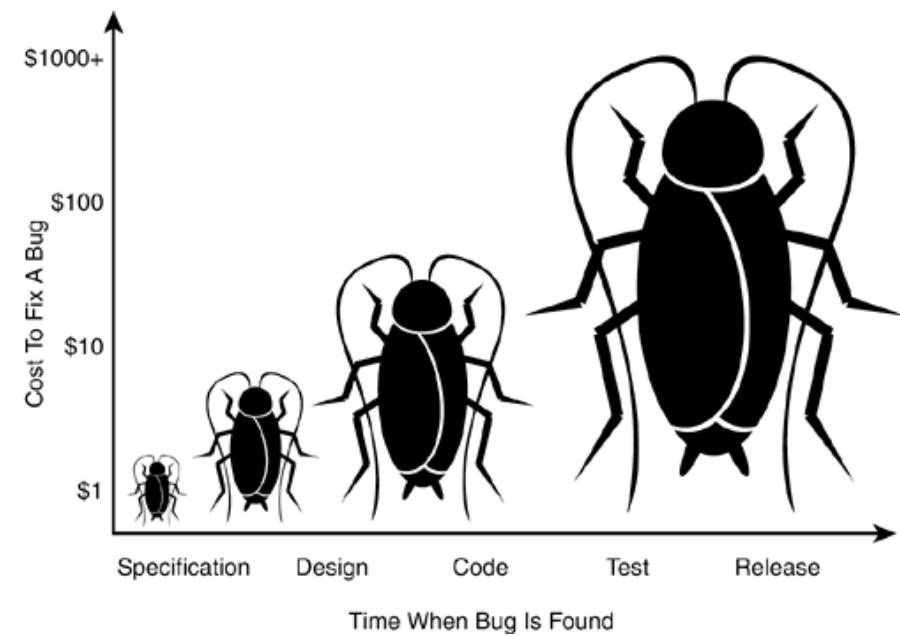
How the customer was billed



How it was supported



What the customer really needed





Bố hay về muộn và
thường xuyên cáu gắt

Bố còn phải fix bug
đến Tết sang năm !

Fix bug triền miên có thể ảnh hưởng đến hạnh phúc gia đình bạn

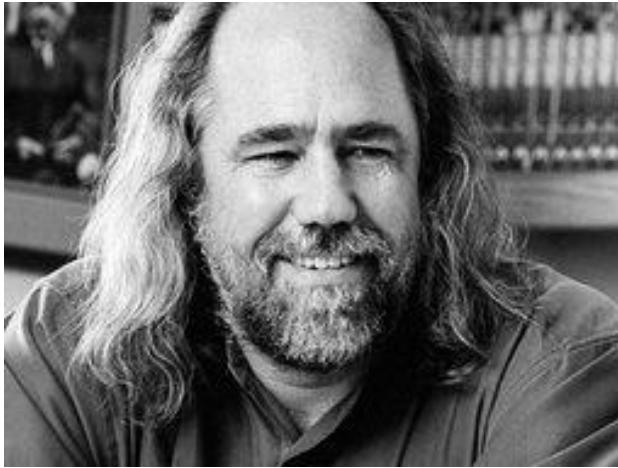
Người nổi tiếng nói gì về Clean Code





Bjarne Stroustrup
30 December 1950
Tác giả C++

I like my code to be elegant and efficient. The logic should be straightforward 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.



Grady Booch
February 27, 1955
Tác giả UML

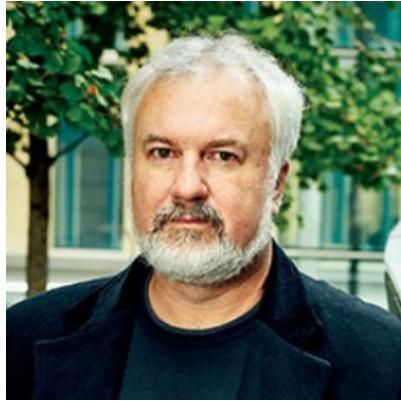
Clean code is simple and direct. Clean code reads like well-written prose. Clean code never obscures the designer's intent but rather is full of crisp abstractions and straightforward lines of control.



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.

David A. Thomas

Tác giả IDE Eclipse.



Michael Feathers
tác giả Working Effectively
with Legacy Code.

Clean code always looks like 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.

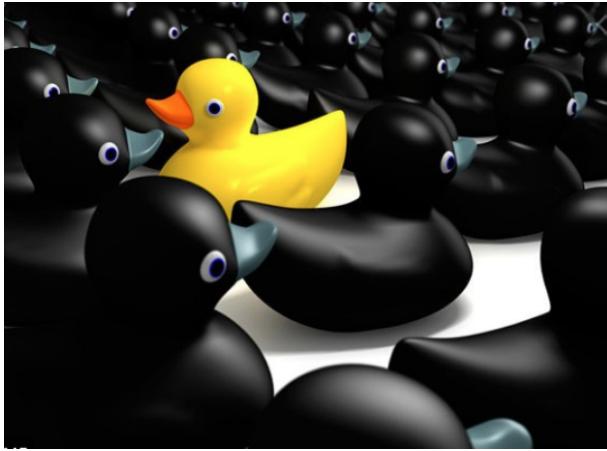
1. Clean code should be elegant, efficient, readable, simple, without duplications, and well-written.
2. You should add value to the business with your code.
3. Clean code offers quality and understanding when we open a class.
4. It is necessary that your code is clean and readable for anyone to find and easily understand

Quy tắc chung – General Rules

1. Follow standard conventions.
2. Keep it simple stupid. Simpler is always better. Reduce complexity as much as possible.
3. Boy scout rule. Leave the campground cleaner than you found it.
4. Always find root cause. Always look for the root cause of a problem.



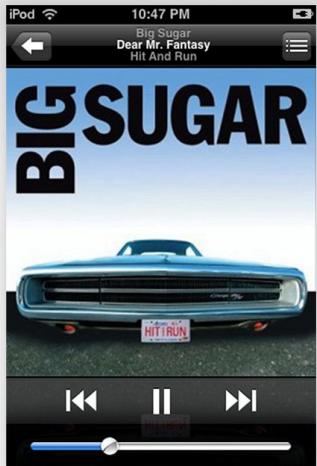
Follow standard conventions



- [Microsoft coding convention](#)
- [Google coding convention](#)
- [JavaScript style guide](#)

Keep it simple stupid

<https://9to5mac.com/2018/07/10/app-store-10-years-design-evolution/>



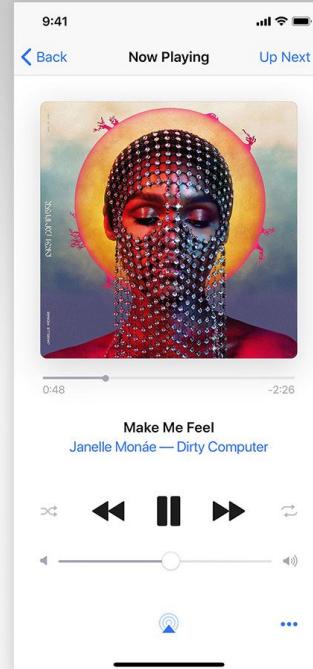
2008
July



2012
November



2013
November



2018
June





2008
July



2008
December



2012
October



2013
September



2017
November

Giao diện ngày càng đơn giản, ít những đường nét 3D

JPA Mapping XML vs Annotation. Which simpler?

```
<entity-mappings>
  <entity class="org.thoughts.on.java.model.Author" name="Author">
    <table name="author" />
    <schema name="bookstore" />
    <attributes>
      <id name="id">
        <generated-value strategy="sequence" generator="author_generator"/>
        <column name="author_id"/>
      </id>
    </attributes>
  </entity>
  <sequence-generator name="author_generator" sequence-name="author_seq" />
</entity-mappings>
```

XML

```
@Entity
@Table(name = "author", schema = "bookstore")
public class Author {
  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "author_generator")
  @SequenceGenerator(name="author_generator", sequenceName = "author_seq")
  @Column(name = "author_id")
  private Long id;
  ...
}
```

Annotation

Boy scout rule – nguyên tắc hướng đạo sinh



Leave the campground cleaner than you found it.

Sau khi cắm trại, phải sạch hơn lúc bạn mới đến

- Code push lên git repository phải tốt hơn, sạch hơn lần trước
- Git repo không phải là nơi lưu code nháp
- Sử dụng **git stash** để lưu nháp khi chuyển sang branch khác

<https://kipalog.com/posts/Su-dung-git-stash-hieu-qua>

Always find the root cause – Tìm và sửa căn nguyên, dừng vá tạm

- Boeing tìm nguyên nhân gây tai nạn hàng loạt máy bay 737 Max
- Samsung tìm nguyên nhân gây Galaxy Note 7 cháy nổ
- Toyota bồi thường 1.1 tỷ USD và thừa nhận lỗi tăng tốc đột ngột
- Apple sửa miễn phí lỗi bàn phím cánh bướm
- Chứng khoán Tokyo gấp lỗi năng chưa từng có, giao dịch ngừng vô thời hạn

Quy Tắc Đặt tên – Naming Rules

Quy tắc đặt tên - Naming Rules

Đặt tên dễ hiểu, tránh nhầm lẫn

Có sự phân biệt rõ ràng

Tìm kiếm được

Dễ phát âm, bình dân

Tiếng Anh phổ thông

Đặt tên dễ hiểu – tránh nhầm lẫn

```
var d; // elapsed time in days
```



```
var elapsedTimeInDays;  
var daysSinceCreation;  
var daysSinceModification;
```



Tên ngắn quá chưa chắc đã dễ hiểu

```
def getData(plist):
    ep = [] // expired person accounts
    for p in plist:
        if p.expiry_date >= datetime.now()
            ep.push(p)
    return ep
```



ep và p là cái gì?

```
def get_expired_persons_list(persons_list):
    expired_persons_list = []
    for person in persons_list:
        if person.expiry_date >= datetime.now()
            expired_persons_list.push(person)
    return expired_persons_list
```

Tên biến dài nhưng
đọc là hiểu

Messages

Gấu dũ

Edit

Anh đang làm gì đây ?
ranh dây không xuống
nhà em chơi .

Anh đang bắn cf nốt
trận anh tới nhé , đang
cầm ACE ^^ .

Em đang coi quán. Đến ngay đi anh, muốn làm roi. À, tiện thể mua báo mới nhé, ở nhà toàn báo cũ thôi. Mà thôi không cần đâu, em vừa mất kính rồi, không nhìn được nữa anh ơi, đến ngay đi, muốn lăm rồi...

Em đang coi quán. Đến ngay đi anh, muốn lăm rồi. À, tiện thể mua báo mới nhé, ở nhà toàn báo cũ thôi. Mà thôi không cần đâu, em vừa mất kính rồi, không nhìn được nữa anh ơi, đến ngay đi, muốn lăm rồi...

Giờ bạn còn muốn dùng tiếng Việt không dấu để đặt tên không?

Tránh những tiền tố, hậu tố thừa

- The (prefix)
- A (prefix)
- Info
- Data
- Variable
- Object
- List
- Array

```
class AuthorInfo          class Author
Person aPerson
Person mot_nguoi           } Person person
Person personObject
getUserInfo();
getClientData();
getCustomerRecord();        } getUser();
```

```
Customer[] customerList; // List 0 thêm thông tin hữu ích  
Table theTable; //Table chứa đối tượng loại gì?
```



```
Customer[] customers; //Chú ý danh từ số nhiều  
Table employees; //Kiểu đối tượng được nêu rõ
```



Tên dễ phát âm, đánh vần

Elon Musk and Grimes' baby's birth certificate
confirms he is legally called X AE A-XII

Lindsay Dodgson

Jun 17, 2020, 4:55 PM



Elon Musk từng là lập trình viên nhưng cũng mắc lỗi khi đặt tên con **X AE A-XII**

Cái nào dễ đọc, dễ nhớ hơn?

```
const yyyyMMddstr = moment().format("YYYY/MM/DD");
```

```
const currentDate = moment().format("YYYY/MM/DD");
```

Tên dễ tìm kiếm. Tránh magic number

```
if (student.classes.length < 7) {...}
```

Magic Numer

```
if (student.classes.length < MAX_CLASSES_PER_STUDENT) {...}
```

Cái nào tốt hơn? Tại sao?



```
setTimeout(blastOff, 86400000); // Không hiểu 86400000 ???
```

// Khai báo constant như dưới, đọc sẽ hiểu được

```
MILLISECONDS_IN_A_DAY = 60 * 60 * 24 * 1000; //86400000; 😊  
setTimeout(blastOff, MILLISECONDS_IN_A_DAY);
```

Refactor đoạn code này thế nào?

```
def create_person(data):
    if data['age']['year'] < 18:
        return None
```

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

Solution

```
def create_person(data):
    if data['age']['year'] < 18: //Magic number !
        return None
```

```
MAX_AGE = 18;
def create_person(data):
    if data['age']['year'] < MAX_AGE:
        return None
    ...
```

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



```
const int MAXCOUNT = 1; //constant, viết hoa tất.  
bool isChanged = true ; //tiền tố is đây là boolean  
interface IRepository //tiền tố I đây là Interface  
public class PersonAddress //Camel case phân biệt từ dễ học  
void getAllOrders() //Camel case nhưng chữ cái đầu viết thường
```

Về cách sử dụng tiền tố prefix để mô tả tính chất biến

Trước đây do công cụ lập trình còn hạn chế ở khả năng gợi ý, không kiểm tra kiểu tức thì, chỉ khi nào biên dịch thì mới báo lỗi, do đó các lập trình thường có những quy ước thêm prefix vào biến để giảm thiểu lỗi khi biên dịch. Nổi tiếng có quy ước Hungarian Notation <http://web.mst.edu/~cpp/common/hungarian.html>

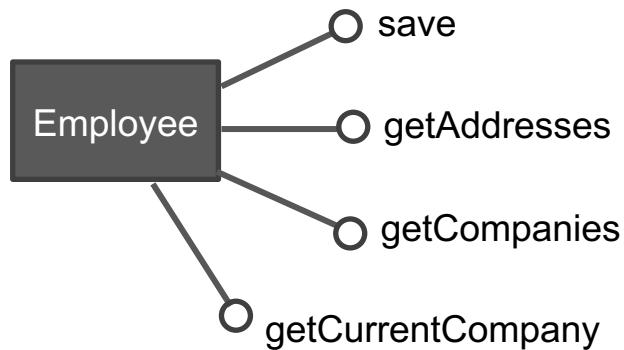
```
int iCount;  
String strName;  
float fSpeed;  
boolean bMale;  
char cACharacter;
```

} Lạm dụng tiền tố khiến tên biến khó đọc

Nay hầu hết ngôn ngữ hỗ trợ tự động nhận kiểu (**type auto inference**) khi được gán. Java, C#, Swift, Dart là **var**, còn C++ là **auto**. Các IDE mạnh hơn VSCode, JetBrains để kiểm tra kiểu trong lúc gõ code.Thêm cả SonarLint phân tích cú pháp nên việc dùng tiền tố prefix đã đi vào dĩ vãng.

Class

- Danh từ – Noun
- Thường là số ít – Singular
`class Person > class People`
- Hãy dùng những từ bình dân, dễ hiểu

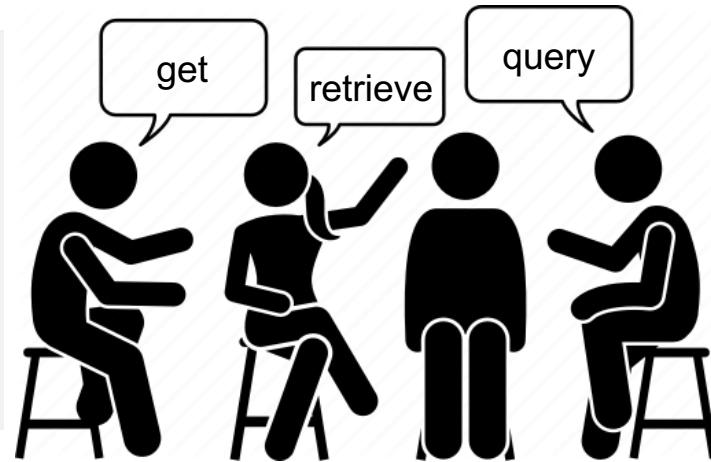


Method, Function

- Động từ - Verb
 - get, retrieve, query
 - take, fetch
 - save, persist
 - flush
 - close, finish
 - discard, abort
 - post, send, notify, push, inform
 - add, edit, update, remove, delete
 - increase, decrease, deduct
 - new, renew
 - deposit, withdraw, refund

Thống nhất tên gọi

```
List<Student> getAllStudents();  
List<Student> retrieveAllStudents();  
List<Student> findAllStudents();  
List<Student> queryAllStudents();  
List<Student> getListOfStudents();
```



- Chọn từ nào bình dân, dễ nhớ, khó viết sai chính tả !
- Dùng nó ở mọi nơi.
- Nếu bạn thấy đồng nghiệp sử dụng một từ đồng nghĩa, hay thảo luận để refactor để cùng thống nhất dùng một từ

```
void loadSingleData()  
-----  
void fetchDataFiltered()  
-----  
void getAllData()  
-----  
void setDataToView()  
-----  
void setObjectValue(int height)
```

```
void getSingleData() ]  
void getDataFiltered() ] Bản chất hành động giống nhau  
void getAllData()  
-----  
void loadDataToView() //nạp dữ liệu vào View  
-----  
void setHeight(int height) //đúng kiểu setter method
```



Tên class và biến chung chung

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



Tên class và biến đọc là hiểu ngay

```
public class ProductWithCategory  
{  
    Entity product;  
    Entity category;  
}
```

```
public class Person
{
    String addressCity;
    String addressHomeNumber;
    String addressPostCode;
}
```



Vi phạm tính đóng gói và khó bảo trì

```
public class Address {
    String addressCity;
    String addressHomeNumber;
    String addressPostCode;
}

public class Person
{
    List<Address> addresses;
}
```



Đóng gói tốt, đúng phạm vi (context)

Có thể tuỳ biến 1 address hoặc nhiều address !

Hàm – Function / Method

- Gọn gàng – làm 1 việc
- Làm đúng tên hàm mô tả
- Tránh tạo ra hiệu ứng phụ
- Không dùng biến cờ rẽ nhánh nhiều việc trong 1 hàm
- Đủ tham số, không thừa, không thiếu



```
public void doSomeThing() {  
    // line 1  
    // line 2  
    // line 3  
    // line 4  
    // line 5  
    // line 6  
    // line 7  
    // line 8  
    // line 9  
    // line 10  
    // line 11  
    // line 12  
    // line 13  
    // line 14  
    // line 15  
    // line 16  
    // line 17  
    // line 18  
    // line 19  
    // line 20  
}
```



95% lập trình viên bị cận
Họ chỉ có thể code font size tối thiểu 14

Màn hình FullHD chứa 20-25 dòng mà không
phải cuộn lên xuống.

Vậy số lượng dòng code trong 1 hàm $\leq 20!$

Hãy refactor hàm này !

```
public void emailClients(List<Client> clients) {  
    for (Client client : clients) {  
        Client clientRecord = repository.findOne(client.getId());  
  
        String regex = "^(.+)@(.+)$";  
        Pattern pattern = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);  
        Matcher matcher = pattern.matcher(clientRecord.getEmail());  
  
        if (clientRecord.isActive() && matcher.matches()) {  
            email(client);  
        }  
    }  
}
```

Vấn đề của hàm emailClients

- Làm quá nhiều việc trong một hàm
- Việc kiểm tra email hợp lệ cần được tái sử dụng vì không chỉ khi gửi mail mà có thể ở phương thức setter email cũng cần kiểm tra.
- Ban đầu điều kiện để gửi email cho từng client đó là clientRecord phải active, email phải hợp lệ, nhưng sau này sẽ bổ xung thêm các điều kiện khác nữa.
- Rồi hàm emailClient sẽ phải chuẩn bị nội dung email, có thể sẽ phức tạp lên

Do đó cần chia nhỏ hàm trên theo từng chức năng một

```
public boolean isValidEmail(String email) {  
    String regex = "^(.+)@(.+)$";  
    Pattern pattern = Pattern.compile(regex);  
    Matcher matcher = pattern.matcher(email);  
    return matcher.matches();  
}
```

Hàm này có thể tái sử dụng ở nhiều nơi

```
public String getClientValidEmail(Client client) {  
    ClientRecord clientRecord = repository.findOne(client.getId());  
    if (clientRecord != null &&  
        clientRecord.isActive() &&  
        isValidEmail(clientRecord.getEmail())) {  
        return clientRecord.getEmail();  
    } else {  
        return null;  
    }
```

Nếu nhiều điều kiện, hãy xuống
dòng cho dễ nhìn

Hàm này làm một việc: nhận vào client và lấy email hợp lệ

Hàm này chỉ làm một việc: duyệt từng client trong danh sách gửi email

```
public void emailClients(List<Client> clients) {  
    for (Client client : clients) {  
        var email = getClientValidEmail(client);  
        if (email != null) {  
            emailTo(email);  
        }  
    }  
}
```

Làm đúng tên hàm mô tả

```
function addToDate(date, month) {  
    // ...  
}  
  
const date = new Date();  
// Khó để biết được hàm này thêm gì thông qua tên hàm.  
addToDate(date, 1);
```



Thêm tháng vào ngày, rõ chưa nhỉ?

```
function addMonthToDate(month, date) {  
    // ...  
}  
  
const date = new Date();  
addMonthToDate(1, date);
```



Bad

- `void doSomethingWithCache()`
- `void addToDate(month, date)`
- `boolean mailValid(email)`
- `customer.getCustomerAddress()`
- `view.reloadTheView()`
- `entityManager.getThingDone()`
- `rawData.preprocessBeforeNextStep()`
- `event.notifyToSomeOne()`
- `void swapAndB(a, b)`

Good

- `void clearCache()`
- `void addMonthToDate(month, date)`
- `boolean isMailValid(email)`
- `customer.getAddress()`
- `view.reload()`
- `entityManager.commitTransaction()`
- `rawData.preprocess()`
- `event.notifyToSubscribers()`
- `void swap(a, b)`

Tránh tạo ra hiệu ứng phụ (side effect)

- Tên hàm cần thể hiện rõ hàm này có tác động, thay đổi dữ liệu (data), trạng thái (state) không.
 - Thay đổi: commit, save, clear, reset, persist, add, update
 - Không thay đổi : isXXX, getXXX, query, retrieve, filter, checkXXX
- Trong hàm không thay đổi (immutable function), **đừng chèn những đoạn mã tác động, thay đổi dữ liệu, trạng thái, tham số truyền vào**

Thay đổi trạng thái trong hàm không thay đổi

```
public boolean checkPassword(String userName, String password) {  
    User user = UserRepo.findByName(userName);  
    if (user != null) {  
        String hashedPassword = BCrypt.hashpw(password, secretKey);  
        String storedHashedPassword = user.getHashedPassword();  
        if (hashedPassword.equals(storedHashedPassword)) {  
            Session.initialize();   
            return true;  
        }  
    }  
    return false;  
}
```

Hàm checkPassword ý nghĩa là một hàm immutable sẽ không thay đổi trạng thái, dữ liệu. Tuy nhiên khi kiểm tra thành công, lại khởi tạo Session.

Đề xuất hãy chuyển `Session.initialize()` ra ngoài

Hàm thay đổi tham số truyền vào khó debug

```
public void wrapInHeading(StringBuilder text) {  
    text.insert(0, "<h1>");  
    text.append("</h1>");  
}
```



Không thay đổi tham số truyền vào, trả về giá trị mới qua return sẽ tốt hơn

```
public String wrapInHeading2(String text) {  
    StringBuilder temp = new StringBuilder(text);  
    temp.insert(0, "<h1>");  
    temp.append("</h1>");  
    return temp.toString();  
}
```



Command Query Separation

```
public Album addSongToAlbum(String songTitle, String albumTitle) {  
    Song song = songRepo.findByTitle(songTitle.toLowerCase());  
    Album album = albumRepo.findByTitle(albumTitle.toLowerCase());  
    album.add(song);  
    return album;  
}
```

```
User user = authenService.login(userID, password);
```

Trong những lệnh Command lại trả về đối tượng sau truy vấn (Query)

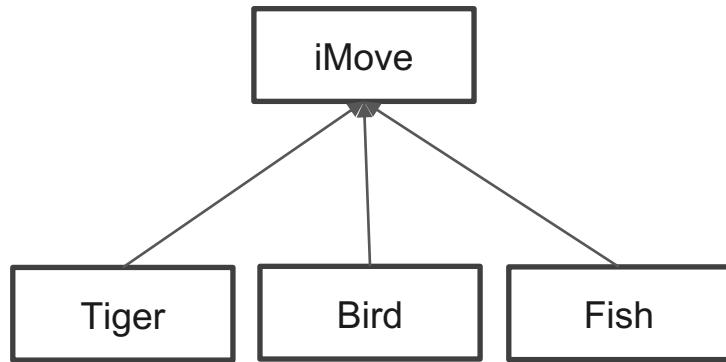


```
public class Animal{  
    public void run() {}  
    public void fly() {}  
    public void swim() {}  
  
    public void move() throws InvalidAnimalType {  
        switch (animal.getType()) {  
            case TIGER:  
                animal.run();  
            case BIRD:  
                animal.fly();  
            case FISH:  
                animal.swim();  
            default  
                throw new InvalidAnimalType(animal.getType());  
        }  
    }  
}
```



Dùng switch để rẽ nhánh logic trong trường hợp này rất khó bảo trì

```
public interface iMove {  
    public void move();  
}
```



Dùng polymorphism sẽ tốt hơn nhiều

```
public class Tiger implements iMove {  
    @Override  
    public void move {  
        //Tiger runs  
    }  
}
```

```
public class Bird implements iMove {  
    @Override  
    public void move {  
        //Bird flies  
    }  
}
```

```
public class Fish implements iMove {  
    @Override  
    public void move {  
        //Fish swim  
    }  
}
```

Không dùng flag arguments để phân nhánh hàm

```
//Trộn 2 logic kiểm thử vào chung một hàm  
public void test(Data data, boolean isSingleTest)
```



Flag argument

```
//Tách ra thành 2 hàm có tên gọi phân biệt sẽ tốt hơn  
public void singleTest(Data data)  
public void multipleTest(Data data)
```



Cách trình bày nào tốt hơn?

```
public void addUser(String fullname, String email, String  
mobile, String address)
```

```
public void addUser(  
    String fullname, //giải thích tham số  
    String email,   //giải thích tiếp  
    String mobile,  //các tham số cẩn lề dễ nhìn hơn  
    String address)
```

Tốt hơn, hàm chỉ có một tham số !

```
public class User {  
    private String fullname;  
    private String email;  
    private String mobile;  
    private Address address;  
    ... }  
public void addUser(User user)
```

Có thể bỏ xung validation logic vào từng trường

Viết logic validate từng trường trong hàm addUser cực kỳ rối và khó bảo trì

Kiểm tra tính hợp lệ tham số truyền vào !

```
/**  
 * @param age must be in range 28 to 85  
 * @return float  
 */  
public float calculatePension(int age) {  
    if (age < 28 && age > 85) {  
        throw new IllegalArgumentException();  
    }  
    ...  
}
```

Hãy dùng Java Doc trong trường hợp sau

- Đây là hàm rất phức tạp, đọc hiểu mất thời gian, chưa kịp refactor
- Cần phải giải thích kỹ giá trị trả về
- Nó là dạng public API của một package sẽ đóng gói binary
- Các tham số đặt tên tương minh nhưng vẫn chưa đủ dễ hiểu



- Java Doc, comment luôn lạc hậu hơn code nên đừng làm dụng quá.
Hãy để code tự giải thích là tốt nhất
- Đừng giải thích hàm đã quá hiển nhiên, và quá ngắn

```
/**  
 * Calculate tax base on shipToCountry  
 * Viết tạm lần tới phải lấy mức thuế trong CSDL chứ không hardcore  
 * US --> 0.07 * Total  
 * EU --> 0.2 * Total  
 * @param total  
 * @param shipToCountry  
 * @return float  
 */  
public float caculateTax(float total, String shipToCountry) {  
    if (shipToCountry.equals("US")) {  
        total += total * 0.07; //US sales tax  
    } else if (shipToCountry.equals("EU")) {  
        total += total * 0.2; //EU VAT  
    }  
    return total;  
}
```



Javadoc Tools

madhavd1.javadoc-tools

Madhav D | ⚡ 8,858 | ★★★★★ | Repository | License | v1.4.0

This extension provides tools to the user for creating Javadocs for classes.

[Disable](#)



[Uninstall](#)

This extension is enabled globally.

[Details](#) [Feature Contributions](#) [Changelog](#) [Dependencies](#)

Javadoc Tools for Visual Studio Code

This extension allows user to generate javadoc comments for all methods within a class. Below commands are available for use -

- **Javadoc Tools: Export Javadoc** - This command allows you to export your Javadoc as well!

Below properties can be set to customize this command

- **javadoc-tools.generateJavadoc.workspaceSourceFolder** - Sets the default source folder which is read when Generating the Javadoc. Default value is the "src" folder in Workspace Root
- **javadoc-tools.generateJavadoc.targetFolder** - Sets the target folder where the Javadoc will be generated. Default path will be \${WorkspaceRoot}\javadoc
- **javadoc-tools.generateJavadoc.runMode** - Set value to run in corresponding mode. Default value is "-public". Possible values are ["-package","-private","-protected","-public"]

- **Javadoc Tools: Generate Comments for Select methods** - This command allows user to choose the methods for which javadoc comments need to be created
- **Javadoc Tools: Generate Javadoc Comments for Open File** - Only generates Javadoc Comments for the open File in focus
- **Javadoc Tools: Generate Javadoc Comments for Workspace** - Generates Javadoc for all classes within the workspace. The files will be opened in the editor and the javadoc comments will be added. This command will not autosave the modified Files