Code đúng liệu đã đủ?

Giới thiệu

goldenskygiang

goldenskygiang

Vũ Trường Giang

- Thành viên ban Kỹ thuật của YouthCode Project
- Lập trình viên C# / ASP.NET Core

Tổng quan

	Nội dung
1	Code đúng liệu đã đủ?
2	Đặt tên – không đơn giản
3	Định dạng sao cho đẹp?

	Nội dung
4	Viết hàm như thế nào?
5	Comment – có cần thiết?
6	Nguyên lý code
7	Code “sạch” trong thi đấu lập trình

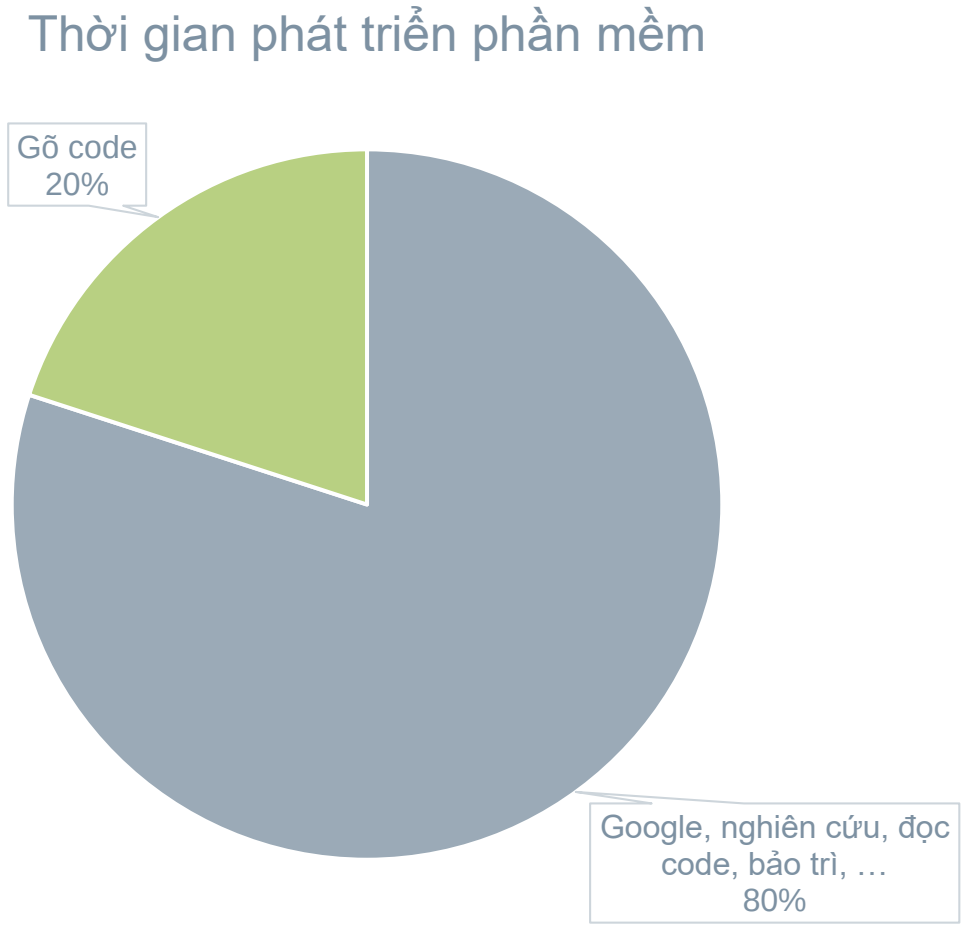
Code đúng liệu đã đủ?

(*) Code đúng: code thực hiện đúng yêu cầu

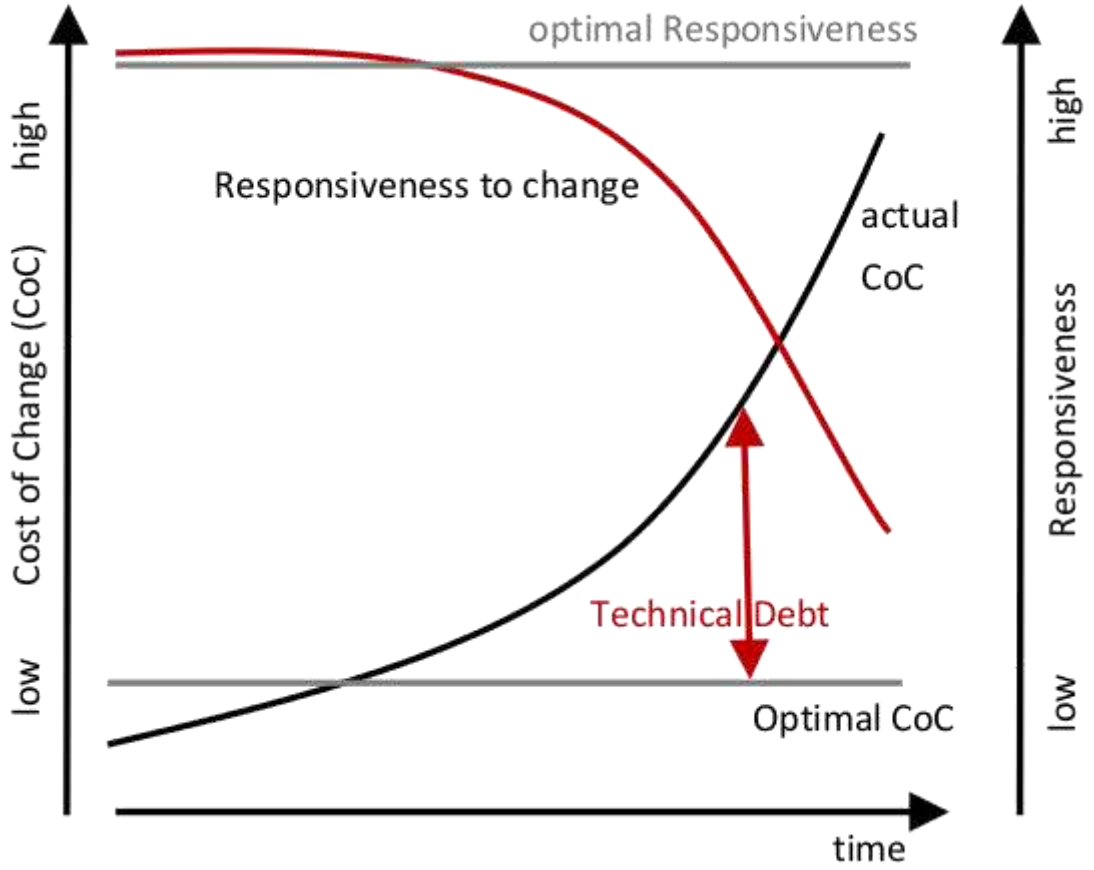
Code đúng liệu đã đủ?

“There are only two hard things in Computer Science: Cache invalidation and naming things.”
– Phil Karlton

Code đúng liệu đã đủ?



Code đúng liệu đã đủ?



Code như thế nào mới đủ?

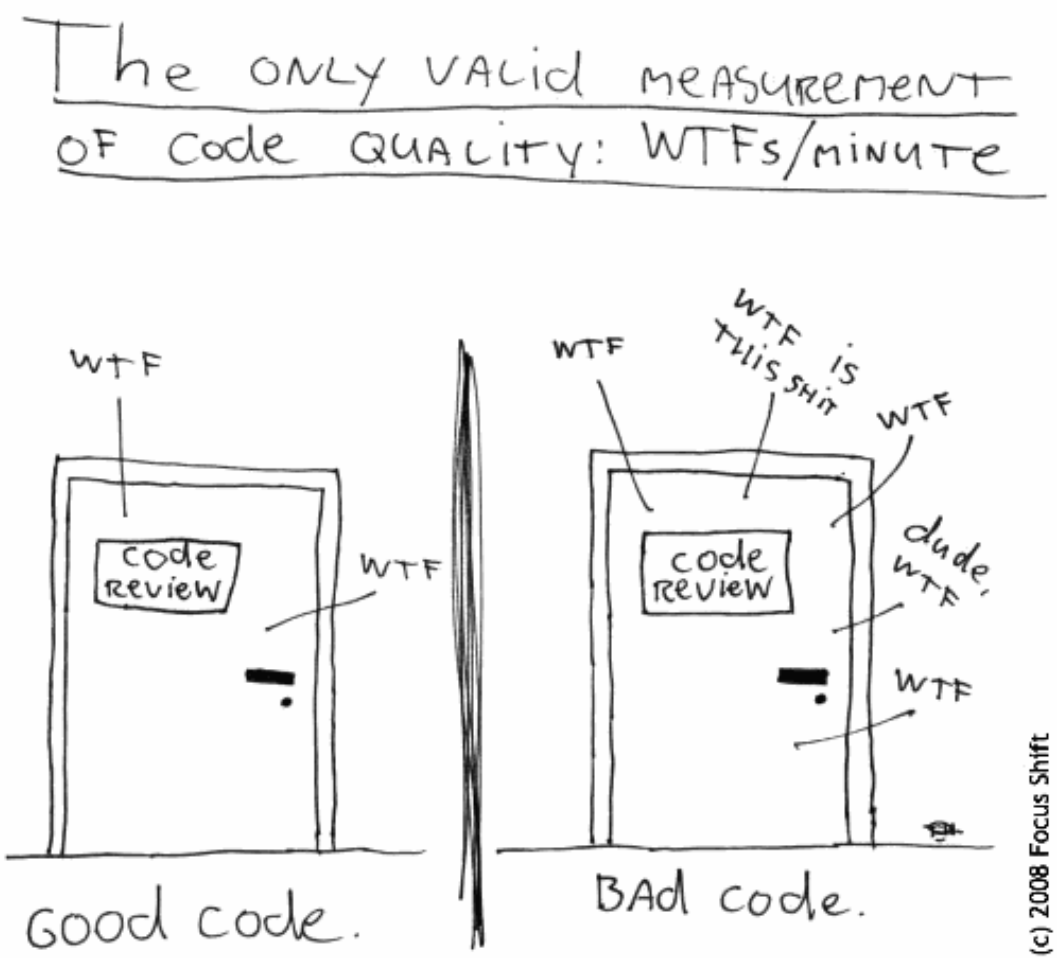
- Code thực hiện đúng yêu cầu.
- Dễ đọc, dễ sửa, dễ bảo trì
- Cấu trúc rõ ràng
- …

CLEAN CODE LÀ GÌ?

Code dễ đọc,

dễ hiểu,

dễ sửa



WTFs/minute: Lower is better

Đặt tên – không đơn giản

Đặt tên có ý nghĩa

› *Tên có mục đích*

- Tốn thời gian đặt tên *nhưng* tiết kiệm thời gian sau này.
- Phải trả lời được:

• Vì sao nó tồn tại?

• Nó làm gì?

• Dùng nó như thế nào?

Đặt tên có ý nghĩa

KHÔNG NÊN

```
var m: array[1..12] of integer =
[31, 28, 31, 30, 31, 30, 31, 31, 30,
31, 30, 31];
    n: integer;
begin
    Write('Nhap n: '); ReadLn(n);
    WriteLn(m[n]);
end.
```

NÊN

```
var daysInMonth: array[1..12] of
integer = [31, 28, 31, 30, 31, 30,
31, 31, 30, 31, 30, 31];
    month: integer;
begin
    Write('Nhap thang: ');
    ReadLn(month);
    WriteLn(daysInMonth[month]);
end.
```

Đặt tên có ý nghĩa

› *Tránh nhầm lẫn*

- Tránh các tên có nghĩa gần giống nhau nhưng khác về dự định.
- Hạn chế dùng các tên dài mà gần giống nhau.

Đặt tên có ý nghĩa

› *Dễ phân biệt rõ ràng*

- Không thể dùng 1 tên để miêu tả 2 thứ phân biệt.
- Nếu 2 tên khác biệt thì chúng phải mang 2 nghĩa khác biệt.
- Dùng danh từ cho tên biến, động từ cho hàm lệnh.
- Không dùng từ đồng nghĩa để làm thay đổi ý nghĩa của biến:

VD: `ProductData – ProductInfo`

Đặt tên có ý nghĩa

› *Dễ phát âm*

- Nếu bạn không phát âm được từ đó, bạn sẽ giống như kẻ ngốc khi nói về nó vậy.

VD: `gnrnTsp` hay `generationTimestamp`

Đặt tên có ý nghĩa

› *Dùng tên dễ tìm kiếm*

- Hạn chế dùng tên biến 1 chữ cái: `i`, `j`, `n`, `m`, …
- Gán tên cho các hằng số được sử dụng nhiều.

Đặt tên có ý nghĩa

› *Tên nên gắn liền với vấn đề/giải pháp*

- Dùng tên thuật toán, thuật ngữ Tin học, …

VD: Tìm kiếm nhị phân: `BinarySearch`, …

- Dùng tên liên quan tới vấn đề đang giải quyết

VD: Bán hàng: `Sales`, `Customer`, `Product`, …

Đặt tên có ý nghĩa

› *Mỗi ý nghĩa một tên*

- Chọn 1 tên cho một ý nghĩa và gắn liền với nó

VD:

`Controller`, `Driver`, `Manager`, ... là các tên cho bộ phận điều khiển

⇨ Chỉ chọn một tên cho bộ phận điều khiển

Đặt tên có ý nghĩa

› *Vì sao đặt tên khó?*

- Đòi hỏi kỹ năng mô tả tốt
- Có kiến thức nền về đời sống

Định dạng sao cho đẹp?

Vì sao cần định dạng code?

- Giao tiếp với đồng nghiệp, giúp họ hiểu code hơn.
- Làm cho code dễ đọc.
- Đặt tiền đề cho việc bảo trì và mở rộng code sau này.

Định dạng code

› *File code nên như thế nào?*

- Tên file ghi rõ tên bộ phận của phần mềm.
- Đọc từ trên xuống dưới, chi tiết của code càng rõ ràng hơn.

VD: Các hàm lệnh theo thứ tự đọc từ trên xuống:

Khởi tạo: `Init()`

Đọc file: `ReadFile()`

Xử lý: `Solve()`

Ghi file: `WriteFile()`

Kết thúc chương trình: `EndExecution()`

Định dạng code

SUNDAY OBSERVER DECEMBER 11, 2005 BUSINESS

Lanka to sign three Bilateral Trade Agreements

BY SUREKHA GALAGODA

Sri Lanka will sign Bilateral Trade Agreements (BTAs) with Egypt, Singapore and USA in the near future said Economic Advisor to President Ajith Nivard Cabraal addressing a workshop on BTAs for South Asian Journalists organised by the Institute of Policy Studies (IPS), UNDP and the South Asian Centre for Economic Journalists.

He said that at present 195 Bilateral Agreements are in force around the globe while the number will increase to 300 by the year 2007. North South Agreements are going beyond trade to include services, labour standards and WTO regulations, he said.

Cabraal said that developing countries are in a weaker position when negotiating a BTA which results in the other country gaining from the weaknesses.

To eliminate the negative effects to the maximum develop a background whereby all stakeholders are consulted before entering into a BTA, Cabraal said.

Director IPS Dr Saman Kelegama said that Multilateral trade agreements existed for ten years in South Asia. First the region introduced SAPTA then SAFTA. BTAs are easy to operate rather than Multilateral trade agreements as they deepen and broaden trade integration.

They are prolifering BTAs not only in the South Asian region but also outside the region.

At present Sri Lanka is considering a BTA with the US and already the TIFA has been signed but in most instances these come at a high cost and they are manifested in different ways such as policy shrinking, high level of IPR protection, commitment to extensively liberalising the services sector and undertakings to commitments to drop monopolies and merger provisions from the competition law.

Former Director, United Nations Conference on Trade and Development B. L. Das said that BTAs are looming large on our economies and constraining our policy space. He said that we go for BTAs to widen our economic prospects. Developing countries are asked to give in areas other than trade but we are not getting the expected market space and policy, he said.

Định dạng code

› *Định dạng theo chiều dọc*

- Một file code ngắn (200 – 500 dòng) thường dễ đọc hơn so với file code dài (> 500 dòng).
- Mỗi dòng một câu lệnh, mỗi nhóm gồm các dòng diễn tả một ý hoàn chỉnh.
- Các ý nên cách nhau một dòng.

Định dạng code

KHÔNG NÊN

```
procedure ReadFile()
begin
    // Reading from file
end;
procedure Solve()
begin
    // Use XYZ algorithm
end;
procedure WriteFile()
begin
    // Write result to file
end;
begin
    ReadFile();
    Solve();
    WriteFile();
end.
```

NÊN

```
procedure ReadFile()
begin
    // Reading from file
end;

procedure Solve()
begin
    // Use XYZ algorithm
end;

procedure WriteFile()
begin
    // Write result to file
end;

begin
    ReadFile();
    Solve();
    WriteFile();
end.
```

Định dạng code

› *Định dạng theo chiều ngang*

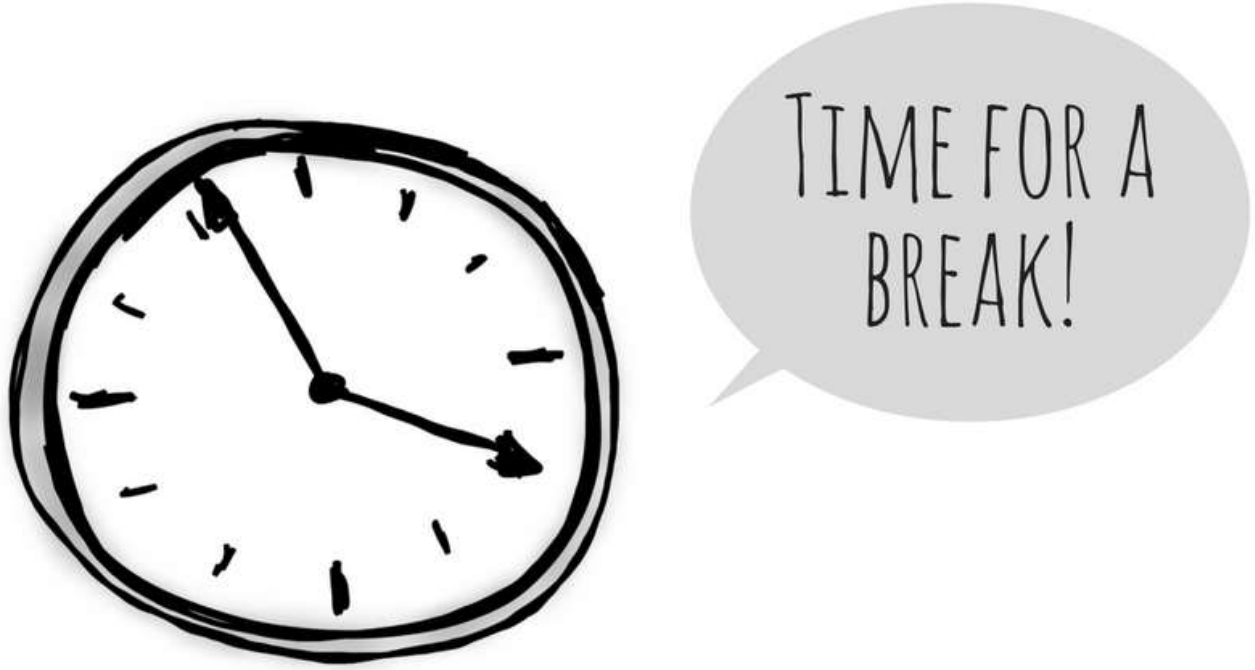
- Dòng code không nên dài quá chiều dài màn hình, hạn chế việc kéo qua kéo lại.
- Dùng khoảng trắng để nhấn mạnh, tách biệt các thành phần trong câu lệnh.
- Thụt lề để phân chia phạm vi hoạt động của code.

Định dạng code

› *Quy định giữa các thành viên trong nhóm:*

- Trước khi lên dự án, phải thống nhất kiểu định dạng code.

TEA-BREAK



Viết hàm như thế nào

Viết hàm như thế nào?

NHƯ THẾ NÀY

```
procedure DoSomething()
begin
    // Random things
end;
```

HOẶC THẾ NÀY

```
function Calculate(): integer;
begin
    // Do some calculations
    // and return the result
end;
```

Viết hàm như thế nào?

› *Nhỏ gọn!*

- Một hàm lệnh không nên dài quá 20 dòng và không quá 150 ký tự/dòng.
- Những khối lệnh `if`, `else`, `while`, … chỉ nên chứa 1 dòng.
- Hàm lệnh không nên quá lớn để chứa các câu lệnh lồng nhau.

Viết hàm như thế nào?

› *Chỉ nên làm MỘT việc!*

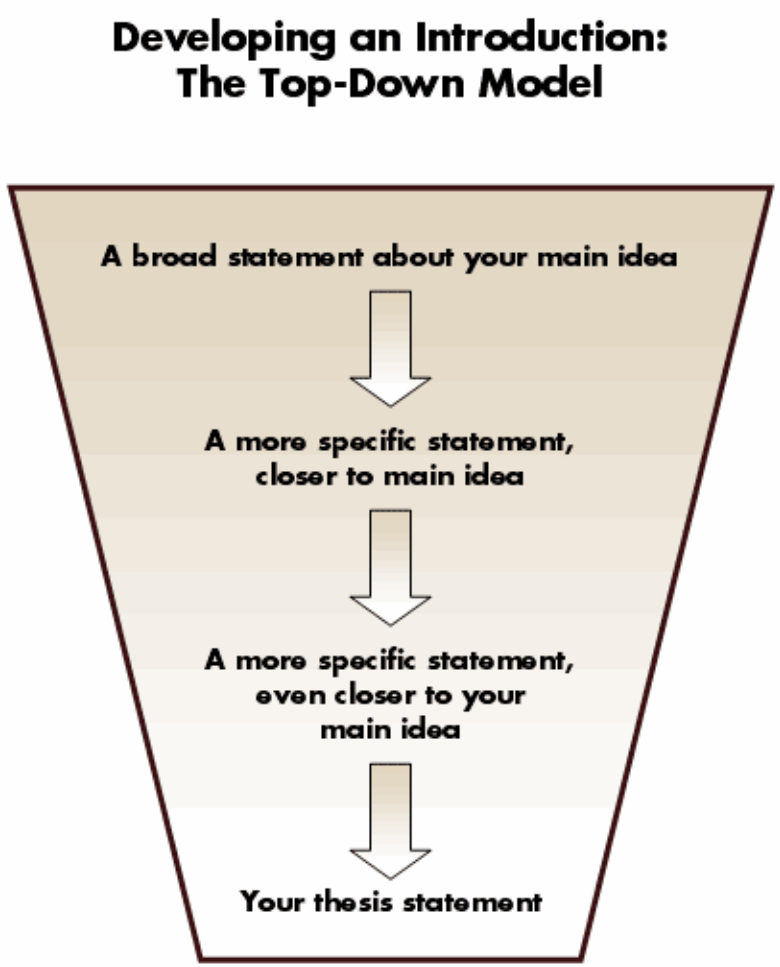
- Mỗi hàm lệnh chỉ nên làm một việc, và nó phải làm tốt việc đó.
- Trong hàm lệnh không được làm những việc khác mà không được nhắc đến trong tên hàm.

Viết hàm như thế nào?

```
function CheckPassword(username, password: string) : boolean;
begin
    if UserManager.IsCorrectPassword(username, password) then
    begin
        UserManager.SignIn(username, password);
        exit(true);
    end
    else exit(false);
end;
```

Viết hàm như thế nào?

› Đọc từ trên xuống dưới



Viết hàm như thế nào

› *Đặt tên hàm*

- Tên hàm nên là động từ.
- Tên dài rõ nghĩa sẽ tốt hơn tên ngắn mà không rõ ràng.

Viết hàm như thế nào?

› *Tránh lặp lại code*

- Ít code hơn, ít lỗi hơn.
- Hạn chế copy & paste code từ chỗ này sang chỗ khác, nên gom lại một nơi.

Comment – có cần thiết?

Comment là gì?

› Là dòng ghi chú và giải thích code

```
// Đây là comment
/* Đây là comment */
{Đây là comment trong Pascal}
# Đây là comment trong Python
```

Comment có cần thiết?

› *Vì sao nó có ích?*

- Giải thích đoạn code
- Dùng để ghi chú
- …

Comment có cần thiết?

› *Vì sao nó có hại?*

- Che giấu đi code xấu.
- Comment thường ít bị thay đổi so với code.

Comment có cần thiết?

› *Nên comment*

- Comment pháp lý, bản quyền.*
- Khi bạn viết thư viện cho người khác sử dụng.
- TODO: những việc cần làm
- Nhấn mạnh điều quan trọng trong code.
- Làm rõ những đoạn code bạn không kiểm soát.

•Nếu comment này lặp đi lặp lại thì nên ghi ra file riêng.

Comment có cần thiết?

› *Không nên comment*

- Khi bạn có thể diễn đạt code tốt hơn.
- Khi code đã rõ ràng.
- Khi comment đó có quá nhiều thông tin thừa.

- Code hỏng hay có vấn đề.
- Ngày tháng, người viết code.

Comment có cần thiết?

› Comment tốt

```
// Don’t run this method unless you have time to kill
procedure SumUpTo10M();
var sum: Int64;
    i: longint;
begin
    sum := 0;
    for i := 1 to 10000000 do
        sum := sum + i;
    WriteLn(sum);
end;
```

Comment có cần thiết?

› Comment xấu

```
// Half all element in an array
function HalfAllElements(arr: TIntArray; size: longint): TIntArray;
var result: TIntArray;
    i: longint;
begin
    for i := 1 to size do
        // Foreach element in array arr, divide that by 2.
        // then save it to the result array.
        result[i] := arr[i] div 2;
    exit(result);
end;
```

Các nguyên lý code

Các nguyên lý code

- Là những kinh nghiệm được đúc kết qua các dự án, thế hệ lập trình viên.
- Tùy thuộc vào hướng lập trình khác nhau, sẽ có những nguyên lý tương ứng.
- Giúp cho code dễ đọc, dễ hiểu, dễ bảo trì.

VD: KISS, DRY, SOLID, YAGNI, …

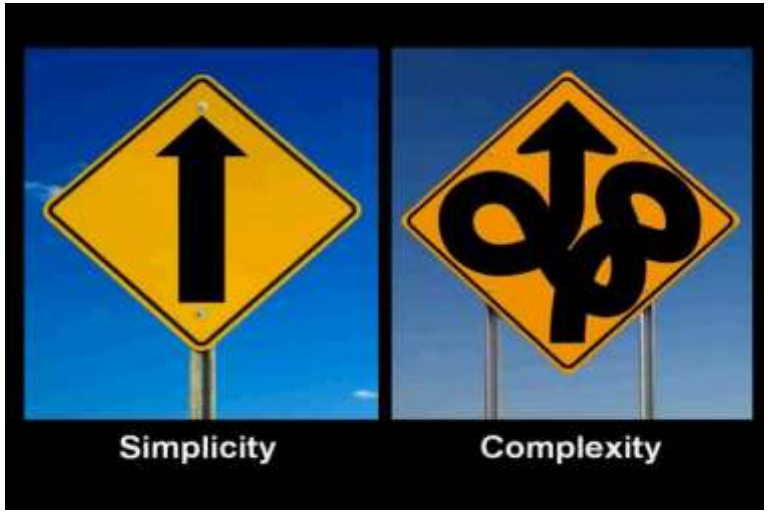
Các nguyên lý cơ bản

› *KISS (Keep It Simple Stupid)*

- Nội dung: “Hãy làm mọi thứ đơn giản lại”.
- Là nguyên lý vàng áp dụng trong rất nhiều lĩnh vực: đời sống, thiết kế, và cả code!

- Vì sao:

- Các vấn đề được giải quyết nhanh hơn, tránh phức tạp.
- Code dễ sử dụng, dễ test.
- Bản thân code chính là tài liệu (self-documented code).
- Dễ dàng bảo trì, sửa lại code khi cần.



Các nguyên lý cơ bản

› *YAGNI (You Ain't Gonna Need It)*

- Nội dung:

"Chưa phải lúc cần thiết thì chưa nên làm"

- Vì sao: Tránh lãng phí thời gian vì

• Tính năng đó có thể không cần thiết.

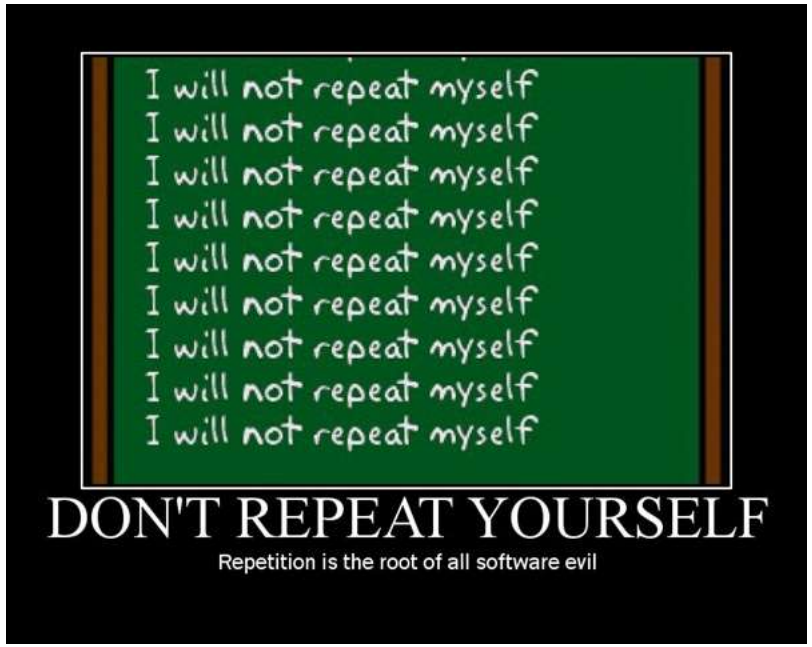
• Tính năng đó có thể không bao giờ dùng tới.

- Chỉ nên tập trung vào công việc hiện tại thay vì thêm cái mới.

Các nguyên lý cơ bản

› *DRY (Don’t Repeat Yourself)*

- Nội dung: “Đừng lặp lại những gì giống nhau”.
- Vì sao:
  - Đơn giản hóa code
  - Tránh lặp đi lặp lại ở nhiều chỗ khác nhau.
- Nếu trong code có các đoạn giống nhau, cố gắng tập trung về một nơi.



Code “sạch” trong thi đấu lập trình

Code “sạch” trong thi đấu lập trình

› *Lúc luyện tập*

- Làm cho code rõ nghĩa, giúp bạn hiểu thuật toán tốt hơn
- Phù hợp cho việc giảng dạy, giúp đỡ

Code “sạch” trong thi đấu lập trình

› *Lúc thi đấu chính thức*

- Không cần thiết phải code “sạch”:

- Bạn chỉ có ít thời gian để giải những bài toán phức tạp.
- Máy chấm đọc bài của bạn.
- Bạn không muốn cho người khác đọc code của mình.

“One difference between a smart programmer
and a professional programmer is that,
the professional understands that clarity is king.
Professionals use their powers for good and write code that others can understand.”
– **Robert C. Martin**

Kết thúc

Cảm ơn mọi người đã chú ý lắng nghe!