II

3 Nguyên tắc cốt loãi để viết clean code

- chọn đúng công cụ cho công việc.

- Tỉ lệ giữ signal và noise (so sách mức tín hiệt tốt và tiếng ồn)

=> Tương tự như vậy clean code sẽ giúp người đọc lọc bỏ phần nhiễu.

- Viết ra những dòng code dễ hiểu mà không cần document hỗ trợ (self-documenting code).

1. Chọn đúng tool.

Khả năng chọn sai tool nằm ở vùng biên và vùng giao giữ các kỹ thuật, ngôn ngữ.

Phải tách biết các loại code chuyên biệt khác nhau: html nên nằm trong file html, css nên nằm trong file css, js nằm trong file js.

- Những lợi ích có được của việc stay native ( giữ đúng bản chất của tool)

+ code của ta sẽ được cached lại.

+ Màu sắc code (code colered)

+ tính năng kiểm lỗi cú pháp (syntax checked)

+ Tách các ngôn ngữ ra từng file giúp cho dễ maintain. (separation of conerns)

+ tính tải sử dụng (reusable)

+ Avoid string parsing (hiệu suất xử lý chuỗi , chỉ riêng cho js)

Nên tránh dùng một ngôn ngữ này dùng chung với ngôn ngữ khác

Avoid using one language to write another language/format via string. E.g. using strings in C#, Java, PHP, etc. to create javascript, xml, json, css

Leverage libraries

One language per file.

2. Maximize signal to noise ration.

Có có tỉ lệ signal to noise cao thì sẽ đễ hiểu hơn. (nếu có quá nhiều tiếng nhiễu, ồn sẽ khó hiểu bài hát).

Mục tiêu của ta là làm tăng tỉ lệ signal to noise.

Signal: đáp ứng được rule TED

Terse: ngắn gọi, súc tích.

Expressive: diễn đạt ý định qua code.

Do one thing: đoạn code chỉ làm một việc hoặc đảm nhiệm một công việc rõ ràng.

Noise: là ngược lại với signal.

Vì người đọc không nhớ được nhiều thông tin cùng một lúc do vậy ta phải chia nhỏ thông tin ra.

Theo tự nhiên thì signal sẽ giảm và noise sẽ tăng do vậy ta phải reflection điều đặng.

II. Principle: Dry principle

Nguyên tắc quan trọng nhất của lập trình là don’t repeat yourself

+ chuẩn hóa cơ sở dữ liệu để không bị trùng lắp thông tin mà người thiết kế DB phải làm

- Các vấn đề liên quan đến việc trùng lập.

+ sao chép đoạn code và dán ở nhiều nơi sẽ làm giảm signal to noise.

+ giảm số dòng code ít nhất có thể, vì càng ít code thì ít bugs phát sinh.

+ việc trùng lặp codes sẽ gây khó khăng trong việc maintain.

II. self- documenting code (viết code sáng sủa, rõ ràng)

Sẽ giúp cho các dev vào sao dễ bắt kịp với team

+ diễn đạt mục đích rõ ràng mà dev đó muốn viết.

+ nên tạo thành nhiều mức độ chi thiết khác nhau.

+ trình bày code hợp lý nhất quán

+ chỉ dùng đến chú thích khi ta không thể diễn tả điều gì đó bằng code.

III. Naming (đặt tên)

1. Hướng dẫn đặt tên class.

- là danh từ

- càng cụ thể càng tốt, có độ gắn kết cao

- signal responsibility

- avoid generic suffixes: tránh thêm những hậu tố không cần thiết.

2. Hướng đặt đặt tên cho method.

Tên method thốt là tên là cho biết chúng làm gì mà người đọc không cần phải đọc code bên trong đó.

Các dấu hiệu cảnh báo đặt tên method làm nhiều hơn một việc.

Các từ and, or, if ví dụ: LoadAndCopy, CalculateAndDisplay.

Tên method không đánh lừa người đọc.

Tránh việc viết tắt method ví dụ RegUsr,

3. Hướng dẫn đặt tên biến.

Tên biến thì nên nghe giống như câu hỏi đúng sai.

Các tên dirty: open, start, status, login.

Các tên tốt (clean) isOpen, done, isActive, loggedln

- khi tên biến chỉ sự đối lập thì dùng các cặp tên biến đối lập.

Ví dụ các cặp tên biến không tốt.

On/disable; quick/slow; lock/ open; slow/ max

Ví dụ các cặp tên biến đối lập tốt.

On/ off; fast/ slow; lock/ unlock; min/ max.

Tóm tắt:

Tên class phải cụ thể không chung chung.

Tên method nên nói những gì có trong method.

Tránh đặt tên gây hiệu ứng lề (side-efects)

Biến Booleans nên như là câu hỏi đúng sai!

Cuối cùng khi gặp khó khăn trong việc đặt tên hãy giải thích chúng thành lời.

IV. Conditionals: conditionals intro, Boolean assignment, positive conditionals.

Dirty:

If(loggedln==true){}

Clean

If(loggedln){}

Hãy sử dụng biểu thức khẳng định thay vì biểu thức phủ định

Thay vì if(!isNotloggedIn) thì nên là if(loggedIn)

IV. Conditions: string comparisons, number comparisons.

Tránh bị stringly type.

Code bẩn: if(employeeType == “manager”

Code sạch: if(employee.Type == EmployeeType.Manager)

1. Strongly typed => No typos ( không có lỗi đánh máy, gõ sai).

2. Intellisense support: có được sự hỗ trợ của intellisense.

3. Document states: biết được tất cả các giá trị của nó (ví dụ ở đây là EmployeeType.)

4. Searchable. Hỗ trợ tìm kiếm.

- Magic numbers

Nên tránh việc sử dụng magic numbers.

Ví dụ code bẩn if(age>21)

Thay vậy thì code sạch sẽ là

Const int legalDrinkingAge = 21;

If(age>legalDrinkingAge){}

IV. Conditionals: complex conditionals.

Theo thời gian thì các điều kiện sẽ phình to ra( có thêm nhiều điều kiện).

Có 2 cách để làm giản đơn điều kiện đó là

+ dùng biến trung gian

+ hoặc dùng hàm trung gian.

Biểu thức điều kiện nên trả lời cho câu hỏi gì.

Trong clean code thì ưu tiên viết code diễn đạt ý đồ hơn là dùng comment, chúng ta chỉ dùng chú thích như là giải pháp sau cùng.

IV. Conditionals: Polymorphism vs Enums, Be Declaretive, Table Driven Methods and Summary.

Tóm tắt

Học được làm rõ mục đích qua các biểu thức điều kiện

Lợi ích của strongly type của hằng và enum.

V. Functions: intro, when to create.

Function và method điều là những đoạn code được gọi bằng tên.

Điểm khác biệt là metho gắn với đối tượng

Function chỉ làm một việc và không nên quá dài.

V. Functions: Avoid duplication, excessive indentation.

Arrow code dùng nói đến những đoạn code lồng nhau hình thành như hình mũi tên. Đó là cho thấy method có độ phức tạp cao, nói cách khác là có nhiều nhãnh rẽ khác nhau bên trong method, làm code khó đọc hơn, khó testing hơn và có xu hướng số bug xảy ra cũng nhiều hơn và người đọc phải nhờ các cấp trong đầu.

- Có 3 cách để loại bỏ arrow code.

+ Extract method (Trích ra method).

Chuyển những dòng code có logic liên quan nhau thành một method và đặt tên rõ nghĩa để giúp truyền tải ý định và khử các cấp lồng nhau.

Khi refrectering thi đi từ dòng code sâu nhất và đi trở ra.

+ fail fast

Ý tưởng là nếu không làm được gì nữa thì hay fail, fail trong trường hợp này nghĩa là ta ném ra một exception ngay khi gặp một tình huốn không mong đợi mà chúng ta không thể xử lý.

+ return early.

Ý tưởng là nếu không có gì để làm nữa thì trở về

V. Functions: mayfly variables, parameters.

Việc khia báo tất cả các biển ở đầu hàm làm người đọc buộc phải ghi nhớ hết tất cả các biến này.

Một function chỉ có những biến myfly.

Myfly là loại côn trùng sống trong thời gian ngắn và do vậy ta cũng nên làm cho các biến có thời gian sống ngắn như vậy.

Có 2 cách để làm việc này:

- Initialize variables just in time: chỉ nên khởi tạo biên khi cần dùng đến.

- Do one thing: tạo những function chỉ làm một việc vì function ngắn thì biến đi vào và đi ra scope cũng rất nhanh.

Bao nhiêu tham số thì hợp lý.

- Ta nên cố gắn chỉ có từ 0 đến 2 tham số, khi ta tập trung vào viết nhưng function nhỏ thì ta đạt được mục tiêu này dễ dàng, việc này làm cho code bạn dễ đọc, dễ hiểu, và dễ test hơn, và cũng đảm bảo function đó chỉ làm một việc.

Những biến kiểu bool trong tham số của hàm cho thấy function này đang làm một lúc 2 công việc trở lên => do vậy ta phải tách chức năng của những function này.

V. Functions: what’s too long? Exceptions.

- nếu function không hiển thị đầy đủ lên màn hình thì nên tách chúng ra.

- khi có nhiều điều kiện trong functions thì nên tách các chức năng đó ra.

- fail slow: Là khi ta khóa ngoại lệ lại và vẫn thi chương trình điều đó dẫn đến các dòng code bên dưới vẫn chạy và dẫn đến chạy sai và rất khó phát hiện được lỗi này.

- nên tách nhóm code ra một hàm rồi bỏ vào lệnh try trong method chứ hàm lúc đầu.

VI. Classes: intro, when to create.

Class với tên gọi tốt sẽ giúp người đọc có cái nhìn sơ bộ về class đó.

When to create

New concept: abstract or real world

Low cohesion: methods should relate

Promote reuse: small, targeted => reuse.

Reduce complexity: solve once, hide away.

Clarify parameters: Identify a group of data.

VI. Class: Cohesion (độ gắn kết)

Là khái niệm để đo các mức độ liên quan giữ các nhiệm của class có chặt chẽ đến đâu.

Class có cohesion cao có nghĩa là enhaces readability, increase likelihood of reuse.

Chúng ta thường mắc sai lầm khi ta giao sai nhiệm vụ cho một class hoặc mong đợi quá nhiều việc từ một class.

Class có cohesion thấp thì rất khó đặt tên vì class có nhiều nhiệm vụ.

Để tránh class có cohesion thấp hãy chú ý các điều sau:

- Hãy chu ý tới các method mà không tương tác tới các thành phần còn lại của class.

- trong class cohesive thì các method và properties kết hợp với nhau để hoàn tất công việc.

- Tương tự cũng chú ý đến các field mà chỉ sử dụng cho một method.

- field nên được sử dụng cho nhiều method trong class và nếu không thì cho thấy đó là dâu hiệu nên tách chức năng sang class khác.

- Nếu một class mà nhận quá nhiều commit hơn bình thường thì class đó là ứng viên để refectering.

Những class có tên là common hay ultility thì thường sẽ tăng số dòng code khá nhanh, tên càng ngắn gọn thì class càng ngắn hơn.

VI. When is a class too small? Primitive Obsession, principle of proximity.

- nếu 2 class dựa vào nhau quá nhiều và gọi phần lớn các method của nhau thì chúng nên gộp lại được thành một class, nếu một class phụ thuộc một class khác quá nhiều thì chúng cũng là một, có thể có quá nhiều class nhỏ sẽ làm cho hệ thống khó hiểu được tổ chức như thế nào.

VI. Outline rule

Outline tốt sẽ làm tăng tỉ lệ signal to noise vì nó cung cấp cho người đọc nhiều lớp trưu tượng, những lớp này được hình thành bằng những method có ý nghĩa, mỗi method ở lớp cao sẽ delegate tới những method ở lớp thấp hơn, bạn có thể có vài method không chứa gì cả ngoài các lệnh gọi đến những method liên quan. Với cách này bạn đã xây dựng được outline trong code của bạn.

Ta nên cố gắn có 3 method ở level cao, khi một class tuân thủ outline ta sẽ dễ tuân thủ theo quy tắc của nó và với nhiều lớp trừu tượng người đọc có thể tìm đúng đến theo thông tin yêu cầu.

VII. Comments: Redundant comment.

Redundant comment là nó lặp lại chính xác những gì mà dòng code kế bên nó thực hiện vì thế chúng không không mang lại hiệu quả gì cả.

Nếu method đã được đặt tên tốt thì ta không cần viết comment cho nó.

VII. Comments: intent, applogies and warnings comments.

Một số cách để tránh tạo ra kiểu comment này là đặt tên function tốt hơn, dùng những biến trung gian với tên gọi ý nghĩa, tạo constant hay enum, refactor biểu thức điều kiện.

Không viêt comment apploly.

Nếu mà đoạn code mà ta chưa fix được hay thêm comment to do cho nó.

Comment warming là cho thấy cần refactor vùng code đó

VII. Comments: zombie code.

Code mà được comment out thì gọi là zombie code.

Làm chậm ứng dụng, gây khó đọc và nếu vì lý do nào đó zombie code sống lại thì ứng dụng của bạn không combine được.

- zombie code làm ta mơ hồ về các dòng code này, cản trở việc refactoring nó cũng tạo noise cho việc tìm kiếm.

Khi bạn comment code thì bạn hãy hỏi

- Có khi nào code này được uncomment hay không?

- Tôi có thể lấy lại chúng từ source control khi cần hay không?

- is this incomplete work that should be worked via a branch?

- is this a feature that should be enable/ disable via configuration?

- Did I refactor out the need for this code?

VII. Comments: Dividers and brace trackers, bloated header, defect log comments.

Khi function quá dài thì developer thường dùng comment để chia tách các phần của chúng.

=> nên bỏ kiểm comment này đi và refactor code ra thành nhiều hàm khác nhau.

Comments – bloated header

- avoid line endings

- Don’t repeat yourself

- Follow conventions

Comments – defect log

Change metadata belongs in source control.

VII. Comments: clean comment and summary.

To do comment là dùng để đánh dấu nơi mà đoạn code mà ta sẽ làm sau này.

Visual studio hỗ trợ cho loại comment TODO này.

Summary comment tốt là phải môt tả code phải làm gì ở mức tổng quát, chúng thường hữu ích cho các class và không nên sử dụng loại comment này cho các tên lớp và tên method không tốt.

Comments – summary

Tôi có thể diễn đạt những gì tôi sắp làm qua code hay không thay vì qua comment?

Tôi đang cố gắng giải thích đoạn code này thay vì refector nó phải không?

Điều này chỉ đơn giản là một thông tin được ghi lại trong source control hay không?

Nếu bạn vừa commet out code hãy nhớ đến những hậu quả của zombie codemay