

OOP Software Metrics

45

45

Mục đích

46

- Hiểu rõ hơn về chất lượng sản phẩm phần mềm
- Đánh giá tính hiệu quả của quy trình
- Cải tiến chất lượng ở 1 phase bất kỳ của dự án

46

Các đặc điểm của độ đo hướng đối tượng

47

- Các độ đo này cần phải cho phép thể hiện được sự khác biệt của các chương trình HDT với các hệ thống khác
- 5 nguyên lý của OOP được sử dụng để xây dựng các độ đo
 - Localization
 - Encapsulation
 - Information hiding
 - Inheritance
 - Object abstraction

47

Localization

48

- Localization: thể hiện việc tập trung thông tin trong 1 chương trình
- Trong ngữ cảnh của OOP, localization thể hiện cả dữ liệu và logic nghiệp vụ đều tập trung trong phạm vi của 1 class
- Vì class là cơ sở trong OOP nên localization thể hiện việc xoay quanh các đối tượng của class cả về thông tin (data) lẫn nghiệp vụ (methods)

48

Encapsulation

49

- Encapsulation thể hiện việc đóng gói (package) các class, các đối tượng với nhau
 - Đối với các chương trình cấu trúc, tính đóng gói có thể được thể hiện qua các cấu trúc (struct) và mảng (collection/array)
 - Đối với các chương trình OOP, tính đóng gói thể hiện các trách nhiệm của 1 class được gắn liền với các thuộc tính, các hành vi của lớp đó

49

Information hiding

50

- Che dấu thông tin thể hiện việc ẩn đi các chi tiết của việc cài đặt 1 class
- Chỉ những thông tin cần thiết mới được cho phép để các đối tượng hoặc thành phần bên ngoài truy cập mới được
- Một hệ thống được thiết kế tốt cần phải làm tốt việc che dấu thông tin
- Information Hiding = chỉ số cho chất lượng của các hệ thống hướng đối tượng

50

Inheritance

51

- Kế thừa là một cơ chế cho phép các nhiệm vụ của 1 đối tượng có thể được sử dụng lại trong các đối tượng khác
- Kế thừa có thể thực hiện ở nhiều cấp
- Rất nhiều độ đo OOP được xây dựng dựa trên nguyên lý kế thừa

51

Abstraction

52

- Tính trừu tượng hoá tập trung vào việc biểu diễn những yếu tố cốt lõi của 1 chương trình/thành phần với ít nhất có thể những chi tiết liên quan đến các tầng thấp (xử lý dữ liệu, xử lý nghiệp vụ...)
- Trừu tượng hoá là một khái niệm có tính tương đối. Chúng ta có thể có các cấp độ trừu tượng hoá khác nhau
- **Mức độ trừu tượng của một lớp là một trong các độ đo hướng đối tượng**

52

Class oriented metrics

53

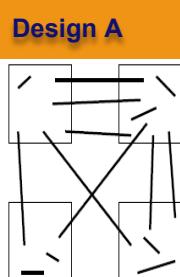
Các độ đo hướng đối tượng bao gồm:

- Chidamber and Kemerer (CK) metrics suite
- Lorenz and Kidd (LK) metrics suite
- The Metrics for Object-Oriented Design (MOOD) Metrics Suite

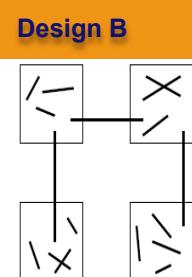
53

What about modularity? Thế nào là tính mô đun hoá của chương trình

- Tính mô-đun hoá là mức độ một hệ thống có thể được chia thành các thành phần khác nhau và kết nối chúng lại với nhau
- Một thiết kế tốt phải dựa trên hai tính chất
 - High cohesion
 - Low coupling



Design A



Design B

- Cohesion: calls inside the module
- Coupling: calls between the modules

	A	B
Cohesion	Lo	Hi
Coupling	Hi	Lo

- Squares are modules, lines are calls, ends of the lines are functions.
- Which design is better?

54

Modularity metrics: Fan-in and Fan-out

Chỉ số về tính mô đun hoá

55

- Fan-in of M: Số lượng lời gọi hàm bên trong mô-đun
- Fan-out of M: Số lượng lời gọi tới M bên ngoài mô-đun M
- Ý nghĩa của Fan-in và Fan-out?
 - Fan-in thể hiện tính gắn kết (cohesion) bên trong mô-đun.
 - Fan-out thể hiện tính kết nối giữa các mô-đun với nhau
 - Một mô-đun càng độc lập nếu lời gọi hàm bên trong mô-đun càng lớn và càng ít lời gọi tới các mô-đun bên ngoài

55

Modularity metrics: Fan-in and Fan-out

Chỉ số về tính mô đun hoá

56

- Modules with fan-in = 0
 - deadcode
 - Nằm bên ngoài phạm vi của hệ thống
- Chỉ số HK của Henry/Kafura cho phép tính mức độ mô-đun hoá của các chương trình thủ tục
 - fan-in: flow of information into procedures
 - fan-out: flow of information out of procedures
 - $HK = SLOC \times (fan-in \times fan-out)^2$
 - SLOC: số dòng code không tính comments và blank lines

56

Chidamber and Kemerer's metrics Độ đo của Chidamber và Kemerer

58

- Số phương thức trên 1 lớp - Weighted methods per class (WMC)
- Chỉ số phản hồi của 1 lớp - Response for a class (RFC)
- Sự thiếu kết dính của các phương thức trong lớp - Lack of cohesion of methods (LCOM)
- Sự kết nối giữa các đối tượng - Coupling between objects (CBO)
- Độ sâu trên cây kế thừa - Depth of inheritance tree (DIT)
- Số lượng các lớp con - Number of children (NOC)

58

Weighted Methods per Class (WMC) Phương thức trên một lớp

59

- WMC tính toán độ phức tạp của một lớp liên quan đến số phương thức của lớp đó hoặc tổng số độ phức tạp của các phương thức trong lớp
 - Độ phức tạp phương thức được tính thông qua chỉ số McCabe của phương thức đó
- Số lượng các phương thức và độ phức tạp của chúng là một chỉ số cho phép xác định thời gian và công sức phải bỏ ra để xây dựng và bảo trì lớp
- Số lượng phương thức của một lớp càng lớn thì tiềm năng ảnh hưởng đến các lớp con càng cao
 - Vì các lớp con kế thừa toàn bộ các phương thức của lớp cha
- Các lớp có số lượng phương thức càng nhiều thì khả năng tái sử dụng tiềm năng của phương thức có thể sẽ không cao nhưng có thể có những ngoại lệ

59

WMC formula

60

```
Class A
{
    Method0 (); // complexity0
    Method1 (); // complexity1
    .
    .
    .
    Methodn (); // complexityn
}
```

$$WMC = \sum_{i=0}^n complexity\ method[i]$$

60

WMC - Example

61

```
public class WMPC {
    public void one() {
        if(true) {
            two();
        } else {
        }
        if(true && !false) {
        }
    }
    public void two() {
        if(true) {}
    }
    public void three(int i){}
}
```

61

Response for a Class (RFC) Phản hồi của một lớp

62

- RFC tính tổng tất cả các phương thức có thể được gọi tới để phản hồi cho một thông điệp tới một đối tượng của lớp hoặc bởi một phương thức nào đó của lớp
 - Chỉ số này cũng tính cho tất cả các phương thức truy cập được trên cây kế thừa của lớp
- RFC xem xét độ phức tạp của một lớp thông qua số lượng phương thức và số tương tác/giao tiếp với các lớp khác
- Số lượng các phương thức được gọi từ một lớp thông qua các thông điệp từ một đối tượng càng lớn thì độ phức tạp của lớp càng lớn
- Nếu một lượng lớn các phương thức có thể được gọi để hồi đáp cho một thông điệp thì việc kiểm thử và sửa lỗi lớp trở nên càng phức tạp bởi vì nó yêu cầu một mức độ hiểu biết lớn hơn về lớp và các trường hợp kiểm thử cần tiến hành

62

Response for a Class (2)

63

- Để tính được RFC:
 - Xác định tất cả các phương thức ở bên ngoài được gọi bởi các methods bên trong class
 - Tạo một tập hợp là hợp của tập hợp các method bên ngoài và các method của class gọi là RS
 - RFC được tính bằng kích thước của RS
- $RS = Mc \cup Me$, $RFC = |RS|$
 - Mc tập hợp các method trong class
 - Me tập hợp các method bên ngoài class được gọi bởi các method bên trong class

63

Example

64

Class: ObjectList

Method:	Insert	Remove	Move	isEmpty	Clear	Find	Show
External Calls:	0	~object	move	0	0	equals	draw

64

Response for a Class(3)

65

- RFC có thể được coi như một độ đo mức độ kết dính và liên kết của class
- High RFC có thể biểu hiện tính coupling cao
- Low RFC có thể chỉ tính coupling thấp
- High RFC có thể dẫn tới
 - Yêu cầu nhiều test cho class
 - Việc hiểu rõ class có thể rất phức tạp
 - Tính tái sử dụng kém
 - Cần nhiều thời gian và nỗ lực bỏ ra để sửa lỗi

65

Lack of cohesion (LCOM) (Sự thiếu độ kết dính)

66

- LCOM đo mức độ không tương tự nhau hay sự khác nhau của các phương thức trong một lớp dựa trên các thuộc tính của lớp (instance variables)
- Các method kết nối với nhau nếu chúng cùng sử dụng thuộc tính của lớp
- LCOM thể hiện độ kết dính của lớp
 - Cùng tương tác trên 1 đối tượng dữ liệu của lớp
- $LCOM(C) = P - Q \text{ if } P > Q \text{ and } = 0 \text{ if otherwise}$
 - P: #pairs of distinct methods in C that do not share variables
 - Q: #pairs of distinct methods in C that share variables

66

Lack of Cohesion in Methods (LCOM) (2)

67

- Một modun có độ kết dính cao có thể tồn tại một cách độc lập
 - Độ kết dính cao thể hiện
 - Việc phân chia lớp tốt
 - Tính đơn giản và tính tái sử dụng
- Sự thiếu độ kết dính hay độ kết dính thấp sẽ làm tăng độ phức tạp của lớp do đó làm tăng khả năng có thể xuất hiện lỗi trong quá trình phát triển phần mềm
- Các lớp có độ kết dính thấp thì có thể chia thành 2 hoặc 3 lớp con để làm tăng tính kết dính trong lớp lên do đó làm tăng tính độc lập của chúng

67

Lack of Cohesion in Methods (LCOM) (3)

69

- Class C with M₁, M₂, M₃
- M₁ = {a, b, c, d, e}
- M₂ = {a, b, e}
- M₃ = {x, y, z}
- P = {(M₁, M₃), (M₂, M₃)}
Q = {(M₁, M₂)}
- LCOM = 1

69

Coupling between object classes (CBO) Sự kết nối giữa các đối tượng của các lớp

70

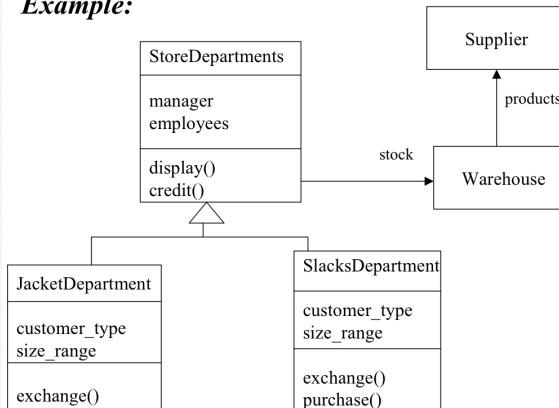
- CBO tính số lượng các lớp có kết nối với lớp được tính
 - Kết nối ở đây được hiểu là số lượng các lớp có liên quan không có trên cây kế thừa của lớp được tính mà lớp đó phụ thuộc vào
 - Sự phụ thuộc ở đây có thể hiểu đơn giản là trong lớp được tính, có gọi đến một phương thức của lớp khác, hoặc trong lớp được tính có một instance object của lớp khác...
 - Bất kì sự liên quan nào đến một lớp khác của lớp được tính cũng được coi như một sự phụ thuộc vào bên ngoài
- Goal: **Lower CBO is better**
 - Tăng tính mô-đun hoá
 - Dễ dàng tái sử dụng

70

Example

71

Example:



71

Coupling between object classes (CBO)

72

- Khi sự liên kết với các lớp khác là quá lớn thì ngăn cản tính mô đun hoá chương trình và ngừa việc tái sử dụng mã nguồn.
- Một lớp càng độc lập (độ phụ thuộc vào lớp khác càng ít) thì nó càng dễ tái sử dụng trong các ứng dụng khác nhau.
- Số lượng các mối liên kết càng lớn thì mức độ nhạy cảm với các thay đổi ở các phần khác nhau của thiết kế càng cao do đó càng khó bảo trì.
- Sự phụ thuộc càng cao thì sẽ làm cho hệ thống càng phức tạp vì một lớp có độ phụ thuộc cao thì sẽ khó hiểu, khó thay đổi và khó sửa chữa.
- Độ phức tạp có thể giảm bằng cách thiết kế các hệ thống với tính lỏng lẻo cao giữa các lớp. Nó làm tăng tính mô đun hoá chương trình và thúc đẩy tính đóng gói cao trong thiết kế.

72

Depth of Inheritance Tree (DIT) Độ sâu cây kế thừa

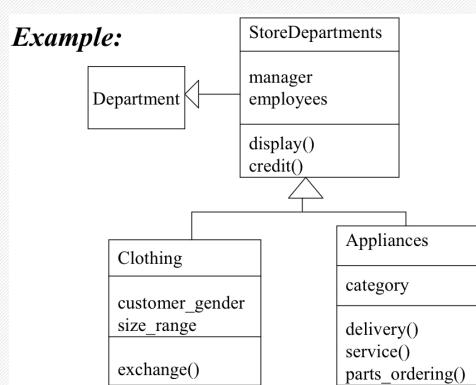
73

- Độ sâu của một lớp trên cây kế thừa được tính bằng số lượng bước tối đa từ nút của lớp đến gốc của cây và được đo bằng số các lớp cha
- Một lớp càng nằm sâu hơn trên cây kế thừa thì số lượng phương thức mà lớp đó được thừa kế sẽ càng nhiều làm cho hành vi của lớp càng phức tạp
- Những cây kế thừa càng sâu thì độ phức tạp thiết kế càng lớn:
 - càng nhiều lớp và phương thức được bao gồm bên trong
 - khả năng tái sử dụng các phương thức kế thừa cũng tăng lên

73

Depth of Inheritance Tree - Example

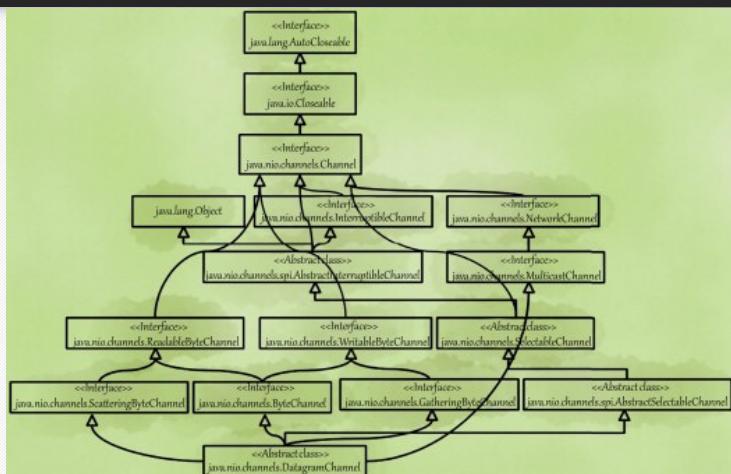
74



74

DIT - Example

75



IT4501 - Software Engineering Department - SolCT/HUST

10/31/22

75

Number of Children (NOC) Số lượng lớp con

76

- Số lượng lớp con NOC là một chỉ số tính số lượng các lớp con trung gian kế thừa từ một lớp (được tính) trên cây kế thừa.
- Nếu một lớp có số lượng con càng nhiều, nó sẽ yêu cầu số lượng ca kiểm thử của các phương thức của lớp đó càng nhiều do đó làm tăng thời gian và chi phí kiểm thử của lớp
- NOC là một chỉ số về độ ảnh hưởng tiềm ẩn của một lớp có thể có đối với toàn bộ hệ thống
 - Số lượng lớp con của một lớp càng lớn thì khả năng cao tính trừu tượng hóa hay tính tổng quát hóa của lớp đó càng giảm
 - Số lượng lớp con càng nhiều thì tính tái sử dụng càng cao bởi vì kế thừa là một hình thức của tái sử dụng trong thiết kế hướng đối tượng

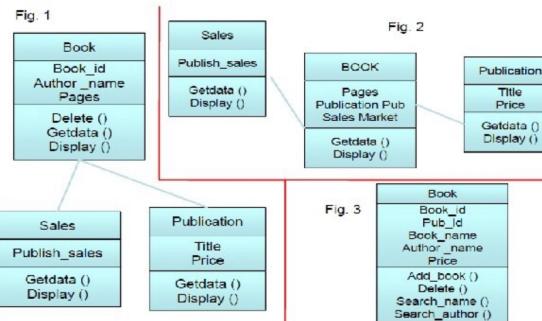
IT4501 - Software Engineering Department - SolCT/HUST

10/31/22

76

Exercise

- Compute WMC, RFC, CBO and LCOM
- Consider complexity to be 1



Package-level Metrics Các độ đo ở mức gói

- Đề xuất bởi Robert Cecil Martin (Uncle Bob)
- Tập trung vào biểu diễn mối quan hệ giữa các package trong một project
- Các độ đo mức gói bao gồm:
 - Khớp nối bên ngoài - Efferent Coupling (Ce)
 - Khớp nối bên trong - Afferent Coupling (Ca)
 - Độ bất ổn định - Instability (I)
 - Độ trừu tượng hoá - Abstractness (A)
 - Khoảng cách chuẩn hoá từ trình tự chính - Normalized Distance from Main Sequence (D)



Why Package?

79

Việc tổ chức code theo các package mang lại hai hiệu quả:

- Kết nối các package với nhau
 - Cho phép chúng ta quan sát thiết kế ở mức độ trừu tượng hoá cao
- Tập trung các class có liên quan trong cùng 1 package giúp tăng độ kết dính của package
 - Cho phép chúng ta dễ dàng sửa đổi và bảo trì

79

Case Study

80



80

Case study

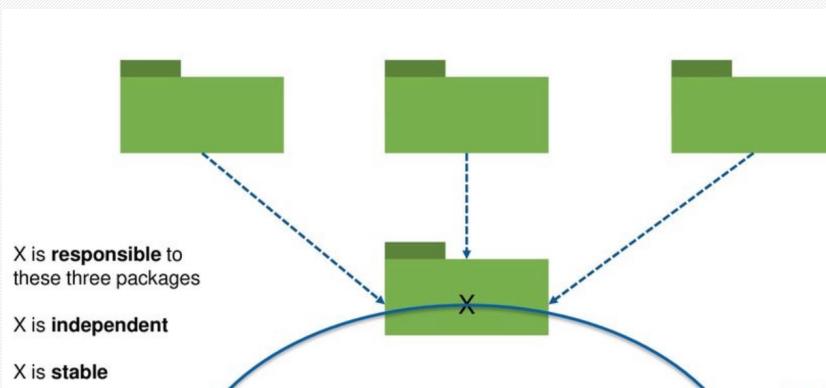
81



81

Stable Package

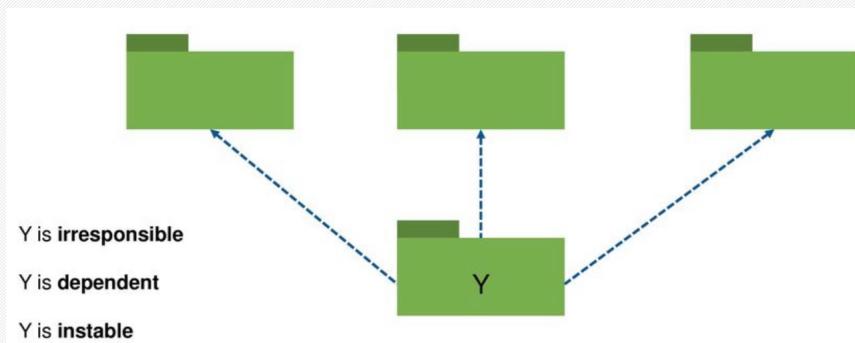
85



85

Instable Package

86



86

“

Depend in the direction of stability

”

Bob Martin

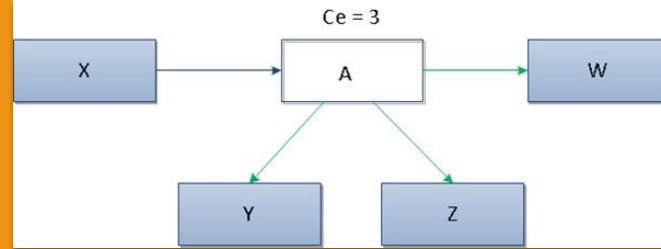
Stable-Dependencies Principle

87

87

Efferent Coupling (CE) (outgoing dependencies)

- CE là độ đo mối quan hệ hướng ra ngoài của một lớp với các lớp khác, nó dựa trên các sự phụ thuộc bên ngoài của các lớp trong một gói
- Độ đo này tính số lượng các lớp trong một gói phụ thuộc vào các lớp của các gói khác



- Giá trị CE cao thể hiện độ không ổn định của 1 gói
 - việc thay đổi của bất kỳ lớp nào nằm bên ngoài gói sẽ yêu cầu các thay đổi trên gói đó
- Giá trị khuyên dùng cho CE là trong khoảng 1 đến 20

IT4501 - Software Engineering Department - SoICT/HUST

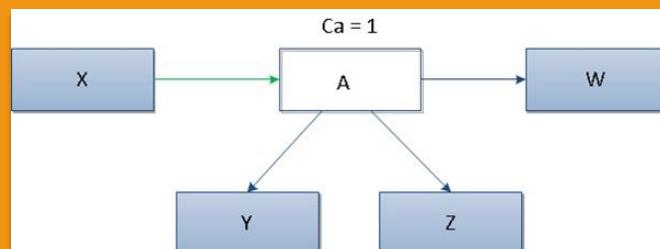
10/31/22

55

88

Afferent Coupling (CA) (incoming dependencies)

- CA – tính toán dựa trên các độ lệ thuộc đến gói - incoming dependencies
- Chỉ số này được tính bằng số các lớp hoặc interface từ những gói khác lệ thuộc vào các lớp trong package được tính



- Chỉ số Ca càng cao thể hiện tính ổn định của gói càng lớn
- Càng nhiều lớp lệ thuộc vào các lớp nằm trong một gói được tính thể hiện tính tái sử dụng của lớp càng cao
- Lớp đó sẽ không thể bị thay đổi một cách dễ dàng vì xác suất thay đổi lan truyền là rất cao vì có quá nhiều lớp lệ thuộc vào lớp đó
- Giá trị thích hợp cho CA của một gói nằm trong khoảng từ 0 đến 500

IT4501 - Software Engineering Department - SoICT/HUST

10/31/22

89

89

Instability (I)

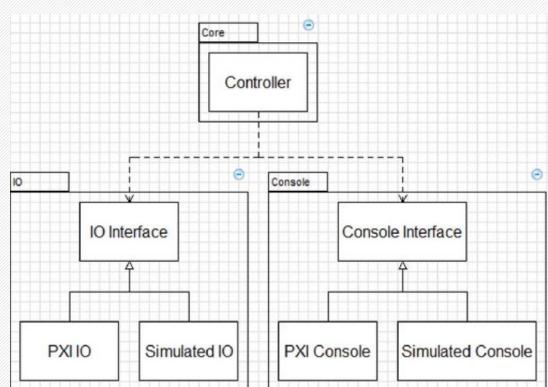
90

- Chỉ số này được sử dụng để đo lường một cách tương đối mức độ nhạy cảm của một lớp đối với thay đổi
- $I = Ce / (Ce + Ca)$
- Đối với một gói hay một thành phần trong hệ thống phần mềm chúng ta có thể chia thành hai loại:
 - Những gói hay thành phần có nhiều phụ thuộc đi (Ce cao) mà có không nhiều phụ thuộc đến (Ca thấp), khi đó giá trị của I nằm gần với 1. Những thành phần này ít ổn định do nó lệ thuộc vào nhiều thành phần khác và khi những thành phần này thay đổi thì nó cũng buộc phải thay đổi theo.
 - Những gói hay thành phần có nhiều phụ thuộc đến (Ca cao) và có ít phụ thuộc đi (Ce thấp), khi đó giá trị của I nằm gần 0. Những thành phần này khó bị thay đổi hơn do trách nhiệm của nó cao hơn.

90

Example

91



91

Stable-Dependencies Principle

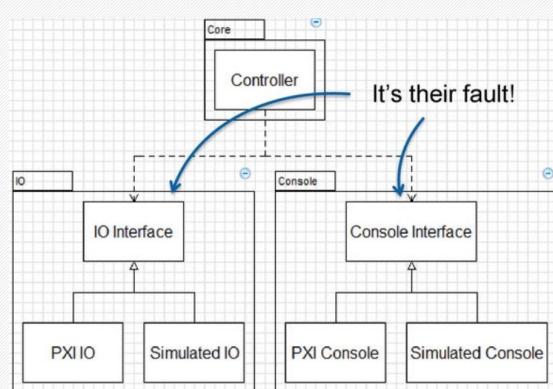
92

- Thay đổi là không tránh khỏi trong quá trình phát triển
 - Một số packages cần phải ổn định
 - Chuyển những phần dễ thay đổi vào các package ở mức cao hơn vì chúng được phép thay đổi
- Các package ở mức trừu tượng cao
 - Không nên phụ thuộc vào các thành phần ở mức thấp hơn (chi tiết hơn)

92

Example

93



Core Metrics

$$\begin{aligned} Ca &= 0 \\ Ce &= 1 \\ I &= 1 \end{aligned}$$

Console Metrics

$$\begin{aligned} Ca &= 1 \\ Ce &= 0 \\ I &= 0 \end{aligned}$$

93

“

A package should be as abstract as it is stable

”

Bob Martin

Stable Abstraction Principle

94

94

Abstractness (A)

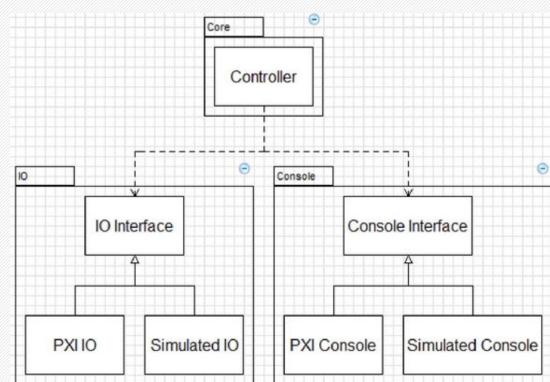
95

- Mức độ trừu tượng hoá A thể hiện mức độ trừu tượng của một gói có phần tương tự với sự bất ổn định gói I
- $A = Ta / (Ta + Tc)$
 - Ta là số lượng các lớp trừu tượng trong gói
 - Tc là số lượng các lớp cụ thể trong gói đó
- Một gói càng ổn định (số đo của I càng gần 0) cũng sẽ có chỉ số A cao ($A \geq 1$)
- Các gói không ổn định ($I \geq 1$) thì sẽ chứa nhiều các lớp cụ thể ($A \leq 0$)

95

Example

96



Core Metrics

$N_a = 0$
 $N_c = 1$
 $A = 0$

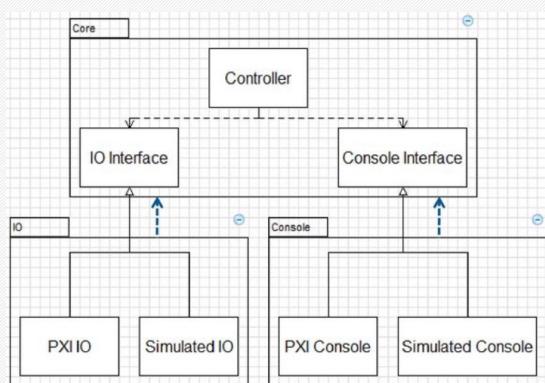
Console Metrics

$N_a = 1$
 $N_c = 2$
 $A = 0.5$

96

Better Solution

97



Core Metrics

$I = 0$
 $A = .667$

Console Metrics

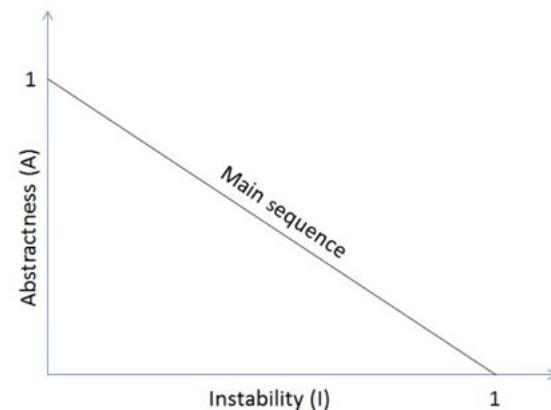
$I = 1$
 $A = 0$

97

Main Sequence Đường trình tự chính

98

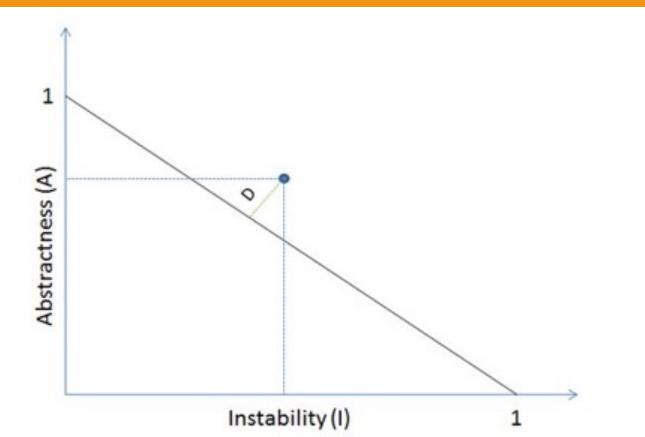
- Sự kết hợp của hai chỉ số A và I cho phép xác định sự tồn tại của trình tự chính của hệ thống như trên đồ thị
- Các lớp được thiết kế tốt sẽ tập trung quanh đường trình tự chính D



98

Normalized distance from main sequence (D)

- Khoảng cách chuẩn hoá từ đường trình tự chính D là một độ đo cho biết mức độ cân bằng giữa tính trừu tượng và tính ổn định của một lớp được tính theo công thức
 - $D = |A + I - 1|$
 - Nếu chúng ta đặt một lớp trên đồ thị đường trình tự chính thì khoảng cách từ nó đến đường trình tự chính sẽ tỉ lệ với giá trị của chỉ số D của nó



99

Benefits of Normalized distance from main sequence (D) Ý nghĩa của chỉ số D

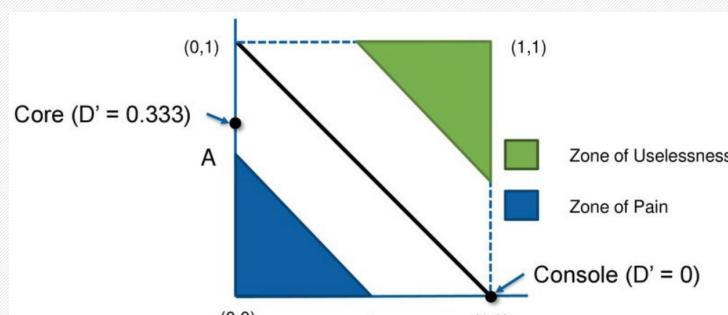
100

- Giá trị của D nên càng thấp càng tốt khi đó thành phần tương ứng (lớp, mô-đun, gói) sẽ càng gần với đường trình tự chính
- Khi $A = 0$ và $I = 0$ thì gói sẽ cực kì ổn định và cụ thể tuy nhiên tình huống đó lại không phải điều chúng ta mong muốn bởi vì gói sẽ rất cứng và khó để có thể mở rộng / thay đổi
- Khi nào $A = 1$ và $I = 1$?
 - tình huống này càng không thể xảy ra bởi vì một gói trừu tượng hoàn toàn sẽ vẫn phải có những liên kết nào đó với bên ngoài sao cho các thể hiện (instance object) cài đặt các chức năng hay phương thức định nghĩa trong các lớp trừu tượng có trong gói có thể được tạo ra

100

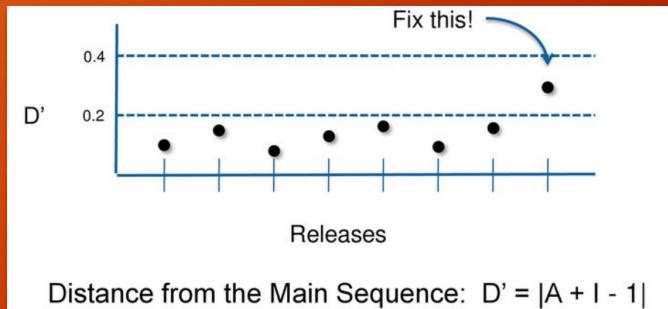
Example

101



101

Track D



102

Summaries

Tổng kết

103

- Hiểu các chỉ số về độ phức tạp mã nguồn, có thể làm các bài tập có liên quan
- Hiểu được ý tưởng và ý nghĩa của các độ đo trong thiết kế hướng đối tượng
 - Phân tích được ý nghĩa của các độ đo
 - Hiểu được làm thế nào để thay đổi chất lượng thiết kế

103

Practice

104

