TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN – ĐHQG TPHCM

---------oOo---------

A picture containing drawing

Description automatically generated

ĐỒ ÁN MÔN HỌC

NGUYÊN LÝ VÀ PHƯƠNG PHÁP LẬP TRÌNH

*Đề tài:*

CÁC TIÊU CHUẨN ĐÁNH GIÁ MỘT NGÔN NGỮ LẬP TRÌNH

Giảng viên hướng dẫn: Ths.Trịnh Quốc Sơn

Lớp: CS111.L21.KHCL

Sinh viên thực hiện:

* Võ Khoa Nam
* Phan Anh Lộc

TP Hồ Chí Minh, ngày 28, tháng 04 ,năm 2021

Mục Lục

[**I.** **GIỚI THIỆU ĐỒ ÁN** 3](#_Toc74932834)

[**II.** **CÁC TIÊU CHUẨN ĐỂ ĐÁNH GIÁ NGÔN NGỮ LẬP TRÌNH** 3](#_Toc74932835)

[Khả năng đọc hiểu (Readability): 4](#_Toc74932836)

[a. Tính đơn giản/tối giản (Simplicity): 4](#_Toc74932837)

[b. Tính trực giao (Orthogonality): 5](#_Toc74932838)

[c. Kiểu dữ liệu (Data types): 6](#_Toc74932839)

[d. Cú pháp thiết kế (Syntax Design): 7](#_Toc74932840)

[Khả năng viết/tính soạn thảo (Writability): 7](#_Toc74932841)

[e. Tính đơn giản và tính trực giao (Simplicity and Orthogonality): 8](#_Toc74932842)

[f. Hỗ trợ cho tính trừu tượng hoá (Support for Abstraction): 8](#_Toc74932843)

[g. Tính biểu đạt (Expressivity): 9](#_Toc74932844)

[Độ tin cậy (Reliability): 9](#_Toc74932845)

[h. Kiểm tra kiểu (Type Checking): 9](#_Toc74932846)

[i. Xử lý ngoại lệ (Handling exception): 9](#_Toc74932847)

[j. Hạn chế truy cập (Restrict aliasing): 10](#_Toc74932848)

[Chi phí (Cost): 11](#_Toc74932849)

[**III.** **Các yếu tố tác động, chi phối đến việc thiết kế ngôn ngữ lập trình:** 11](#_Toc74932850)

[Kiến trúc máy tính: 11](#_Toc74932851)

[Các phương pháp thiết kế lập trình: 12](#_Toc74932852)

[**IV.** **So sánh các ngôn ngữ lập trình:** 13](#_Toc74932853)

[**V.** **Kết luận:** 15](#_Toc74932854)

# **GIỚI THIỆU ĐỒ ÁN**

Ngôn ngữ lập trình không khác lắm so vơi ngôn ngữ nói. Học bất kỳ ngôn ngữ nào đòi hỏi sự hiểu biết về các khối xây dựng và ngữ pháp chi phối việc xây dựng câu lệnh trong ngôn ngữ đó. Vì vậy khi phát triển các ứng dụng hay phần mềm, một lập trình viên luôn phải thích ứng với nhiều ngôn ngữ, cấu trúc, đặc điểm của ngôn ngữ lập trình mà họ đang sử dụng. Những lập trình viên khi sử dụng ngôn ngữ lập trình, thường có những quan điểm, những nhận định mang tính chủ quan của mình về ngôn ngữ lập trình mà họ đang sử dụng. Tuy nhiên, nội dung mà cộng đồng quan tâm đó là những tiêu chuẩn nhận xét, đánh giá một cách khách quan, được nghiên cứu sâu và được công nhận trên các bài báo, trên các diễn đàn lớn hay là được chuyển hoá thành sách. Vì vậy thông qua đồ án này chúng tôi giúp những lập trình viên còn mơ hồ về cách đánh giá ngôn ngữ lập trình, có một cách nhìn đúng đắn, chính xác hơn về ngôn ngữ lập trình, thông qua các nghiên cứu mà chúng tôi tổng hợp được.

# **CÁC TIÊU CHUẨN ĐỂ ĐÁNH GIÁ NGÔN NGỮ LẬP TRÌNH**

Để đi đánh giá một ngôn ngữ lập trình cần xây dựng một bộ tiêu chí đánh giá. Một danh sách các tiêu chí như vậy nhất thiết phải gây tranh cãi bởi vậy rất khó để khiển các nhà khoa học đồng ý về giá trị của một số đặc tính ngôn ngữ nhất định so với những đặc tính khác. Bất chấp những khác biệt này, hầu hết đều đồng ý rằng các tiêu chí được đề cập ở đây là quan trọng.

Có bốn tiêu chí chính để đánh giá một ngôn ngữ lập trình: khả năng đọc (Readability), khả năng viết (Writability), độ tin cậy (Reliability) và cuối cùng là chi phí (Cost).

**Hinh 1: Các tiêu chuẩn chính đánh giá ngôn ngữ lập trình**

A picture containing table

Description automatically generated

*Hinh : Các tiêu chuẩn để đánh giá ngôn ngữ lập trình*

Một số đặc điểm ảnh hưởng đền ba trong bốn tiêu chí quan trọng nhất này được thể hiện ở Hình 1, tiêu chí cơ bản thứ tư là chi phí (cost) không được đưa vào bảng vì nó không ảnh hưởng nhiều đến các tiêu chí khác và các đặc điểm trong Hình 1.

## Khả năng đọc hiểu (Readability):

Một trong những tiêu chí quan trọng nhất để đánh giá một ngôn ngư lập trình là tính dễ đọc và hiểu của chương trình. Trước những năm 70, khi tài nguyên và bộ nhớ máy tính bị hạn chế, lập trình dựa trên tính hiệu quả. Phát triển phần mềm dựa trên việc viết mã, đặc điểm tích cực chính của ngôn ngữ lập trình là tính hiệu quả, các cấu trúc ngôn ngữ được thiết kế theo quan điểm của máy tính hơn là của người dùng máy tính [1],[3].

Tuy nhiên, trong những năm 70, có một khái niệm mới được cho ra đời đó là khái niệm vòng đời phần mềm (software lìe-cycle, Booch,1987), với khái niệm này phần mềm có vòng đời phát triển, được liên kết theo các giai đoạn khác nhau. Bảo trì được coi là một phần chính của chu trình, đặc biệt là về mặt chi phí. Bởi vì tính dễ bảo trì được xác định một phần lớn bởi khả năng đọc của các chương trình, tính dễ đọc đã trở thành một thước đo qua trọng để đánh giá chất lượng của chương trình và ngôn ngữ lập trình. Có sự giao thoa khác biệt từ việc tập trung vào định hướng máy móc sang tập trung vào định hướng con người [1],[3].

Khả năng đọc phải được xem xét trong bối cảnh của miền vấn đề (problem domain). Nếu một chương trình giải quyết vân đề được viết bằng ngôn ngữ không được thiết kế để giải quyết vấn đề đó, thì chương trình đó có thể không tự nhiên và phức tạp, gây khó đọc một cách bất thường [1],[3]. Ví dụ như chúng ta sử dụng ngôn ngữ BareBones để viết chương trình biểu diễn thuật toán tìm kiếm nhị phân thì chương trình có thể không đáp ứng được yêu cầu (vì chỉ có 4 câu lệnh), hoặc là mất nhiều thời gian và thao tác sẽ dài dòng hơn là chúng ta sử dụng những ngôn ngữ như C++ hoặc Python để cài đặt.

Khả năng đọc là khả năng hiểu những gì đã được viết, được xác định

dựa trên một số đặc điểm PL [32] và các đặc điểm của ngôn ngữ chuyển đổi. Việc đánh giá khả năng đọc của một ngôn ngữ phải được thực hiện trong bối cảnh ứng dụng. Chính xác hơn, so sánh khả năng đọc của hai ngôn ngữ với

các miền ứng dụng khác nhau là không hợp lý. Các đặc điểm và tính năng của

khả năng đọc được mô tả chi tiết trong phần sau.

### Tính đơn giản/tối giản (Simplicity):

Một ngôn ngữ lập trình đơn giản và dễ đọc chắc chắn sẽ được đánh giá cao hơn một ngôn ngữ nhiều cấu trúc, gây phức tạp, khó hiểu và trên hết là nhàm chán hơn. Tính đơn giản của một ngôn ngữ lập trình ảnh hưởng mạnh mẻ đến khả năng đọc hiểu của chương trình. Một ngôn ngữ có nhiêu cấu trúc cơ bản sẽ khó đọc hơn là một ngôn ngữ ít cấu trúc hơn. Các lập trình viên sử dụng một ngôn ngữ lớn thường học một tập hợp con của ngôn ngữ và bỏ qua các tính năng khác của nó. Lấy ví dụ về nạp chồng toán tử, mặc dù nó là một khả năng rất hữu ích trong một ngôn ngữ, nhưng nó có thể gây nhầm lẫn nếu người đọc không biết được cách nạp chồng của nó đã được thực hiện như thế nào. Nhưng các vấn đề vê tính đọc hiệu chỉ diễn ra khi người viết sử dụng tập con các cấu trúc mà người đọc không quen thuộc [1],[3].

Đặc điểm phức tạp thứ hai của ngôn ngữ lập trình là tính đa dạng (feature multiplicity). Tính đa dạng được hiểu đơn giản là có nhiều cách thực hiện một thao tác cụ thể [1],[3]. Ví dụ, trong ngôn ngữ C++ và Python các thao tác tăng giá trị của một biến số nguyên được thực hiện bằng nhiều cách khác nhau

**Hinh 2: Thao tác tăng giá trị của biến Count khác nhau trong các ngôn ngữ lập trình**

Count = Count + 1 ++Count

Count += 1

Count++

*Hinh : Ví dụ minh hoạ về phép tăng giá trị của một biến trong python và C++*

Mặc dù hai câu lệnh Count++ và ++Count (tăng sau và tăng trước) có ý nghĩa hơi khác nhau và với những câu khác trong một số ngữ cảnh, tất cả chúng đều có cùng ý nghĩa khi được sử dụng làm biểu thức độc lập.

Một vấn đề tiềm ẩn thứ ba là nạp chồng tán tử (operator overloading), trong đó có một ký hiệu toán tử đơn có nhiều hơn một ý nghĩa. Mặc dù đều này thường rất hữu ích, nhưng nó có thể dẫn đến giảm khả năng đọc nếu người dung được phép tạo ra toán tử của riêng học và không làm việc đó một cách hợp lý [1],[3].

Tất nhiên sự đơn giản trong ngôn ngữ lập trình có thể bị mang đi quá xa. Ví dụ như các câu lệnh hợp ngữ Assembly là mô hình của sự đơn giản. Tuy nhiên, chính sự đơn giản này làm cho các chương trình hợp ngữ khó đọc hơn. Bởi các câu lệnh điều khiển phức tạp hơn, cấu trúc chương trình ít rõ ràng hơn. Các câu lệnh đơn giản thường được yêu cầu nhiều hơn so với các câu lệnh trong ngôn ngữ cấp cao [1],[3].

### Tính trực giao (Orthogonality):

Định nghĩa về tính trực giao hơi rườm rà: tính trực giao là khả năng kêt hợp các cấu trúc nguyên thuỷ của một ngôn ngữ, theo một số cách nhỏ, để xây dựng dữ liệu và cấu trúc điều khiển và dữ liệu của ngôn ngữ [1],[3]. Trong ngôn ngữ C++ có hỗ trợ bốn kiểu dữ liệu nguyên thuỷ (int, float, char, double) và hai toán tử kiểu( array và pointer). Nếu hai toán tử kiểu có thể được áp dụng cho chính chúng và bốn kiểu dữ liệu nguyên thuỷ, thì một số lượng lớn cấu trúc dữ liệu có thể được xác định.

Một ngôn ngữ là trực giao khi một tập hợp nhỏ các cấu trúc cơ bản có thể được kết hợp theo một số cách tương đối nhỏ để tạo cấu trúc dữ liệu của ngôn ngữ. Mọi sự kết hợp có thể có của số lượng nguyên thủy tương đối nhỏ nên được cho phép bằng ngôn ngữ trực giao và có cấu trúc có ý nghĩa.

Nếu một ngôn ngữ thiếu đi tính trực giao, thì ngôn ngữ đó sẽ bị giới hạn. Ví dụ, giả sử trong tình huống bạn sử dụng hai toán tử khác nhau để tính tổng các số nguyên hoặc các số thực. Phép toán tổng là giống nhau về mặc khái niệm, nhưng chúng ta không thể sử dụng một toán tử trên các cấu trúc khác nhau (int hoặc float) [1],[3].

Tính trực giao thường liên quan chặt chẽ đến tính đơn giản. Thiết kế ngôn ngữ càng trực giao thì càng ít ngoại lệ mà các quy tắc ngôn ngữ yêu cầu. Ít ngoại lệ hơn có nghĩa là mức độ đều đặn cao hơn trong thiết kế, giúp ngôn ngữ dễ học, đọc và hiểu hơn.

Tuy nhiên, tính trực giao quá nhiều cũng có thể gây ra vấn đề, làm cho ngôn ngữ trở nên phức tạp ở mức không cần thiết. Lấy ví dụ về ngôn ngữ trực giao nhất là ALGOL 68. Mỗi cấu trúc trong ALGOL 68 đều có một kiểu và không có giới hạn nào đối với các kiểu đó. Sự tự do tổ hợp này tạo ra các cấu trúc cực kỳ phức tạp [1],[3].

**Hinh 3: Ví dụ về cấu trúc trực giao trong ngôn ngữ ALGOL 68 [2].**

**int** n = 2;

**co** n is fixed as a constant of 2. **co**

**int** m := 3;

**co** m is a newly created local variable whose value is initially set

to 3. co

**co** This is short for ref int m = loc int := 3**; co**

real avogadro = 6.02214; co Avogadro's number **co**

*Hinh : Ví dụ minh hoạ đoạn code ngôn ngữ ALGOL 68*

### Kiểu dữ liệu (Data types):

Việc định nghĩa kiểu dữ liệu và cấu trúc dữ liệu một cách rõ ràng trong một ngôn ngữ lập trình sẽ là một sự trợ giúp quan trọng cho khả năng đọc hiểu của chương trình. Ví dụ, ta có câu lệnh sau

TimeOut = 1

Ta muốn dùng biến trên như một biến kiểm tra kiểu đúng sai, ta sẽ sử dụng kiểu Boolean để định nghĩa biến đó để việc sử dụng dễ dàng hơn.

TimeOut = True

### Cú pháp thiết kế (Syntax Design):

Các cú pháp, hoặc hình thức, các yếu tố của một ngôn ngữ có ảnh hưởng đáng kể đến khả năng đọc của chương trình.

Có hai lựa chọn thiết kế cú pháp ảnh hưởng đến khả năng đọc:

* Các từ đặc biệt: sự xuất hiện của chương trình và do đó khả năng đọc của chương trình bị ảnh hưởng mạnh mẽ bởi các hình thức của các từ đặc biệt trong ngôn ngữ lập trình( ví dụ: while, for, class, vector,…). Đặc biệt là phương pháp hình thành câu lệnh ghép, hoặc nhóm câu lệnh, chủ yếu trong cấu trúc điều khiển. Một số ngôn ngữ đã dùng cặp từ hoặc ký hiệu đặc biệt để bắt đầu và kết thúc một nhóm lệnh. Ví dụ, trong ngôn ngữ C, C++ có sử dụng dấu ngoặc nhọn ( { } ) để chỉ định các câu lệnh ghép. Hay trong pascal việc định nghĩa các câu lệnh ghép sẽ được đặc trong cặp từ (begin, end). Trong Fortran 95 và Ada làm điều này rõ ràng hơn bằng cách sử dụng cú pháp đóng riêng biệt cho từng loại nhóm câu lệnh. Ví dụ, trong Ada sử dụng “end if” để kết thúc cấu trúc lựa chọn, hay end loop để kết thúc cấu trúc vòng lặp [1],[3].
* Một vấn đề quan trọng khác là liệu các từ đặc biệt của ngôn ngữ có thể được sử dụng làm tên cho các biến chương trình hay không. Nếu vậy, các chương trình kết quả có thể rất khó hiểu. Ví dụ trong ngôn ngữ Fortran 95, các từ đặc biệt như Do va End được xem như tên biến hợp lệ [1],[3].
* Hình thức và ý nghĩa: thiết kế các câu lệnh sao cho sự xuất hiện của chúng ít nhất cũng cho thấy được ý nghĩa, mục đích và chức năng của nó. Để người đọc có thể tiếp cận và dễ dàng áp dụng hơn. Một trong những phàn nàn cơ bản về các lệnh shell của UNIX (Raymond, 2004) là không phải lúc nào cũng gợi ý cho người dung các chức năng của các câu lệnh đó. Ví dụ, ý nghĩa của lện Grep UNIX là lệnh sẽ tìm xem một chuỗi nào đó có tồn tại trong file hay không. Nếu như một người bình thường không trang bị kiến thức về lập trình shell UNIX sẽ không thể hiểu được ý nghĩa của lệnh đó [1],[3].

## Khả năng viết/tính soạn thảo (Writability):

Khả năng viết là thước đo mức độ dễ dàng sử dụng của một ngôn ngữ để tạo các chương trình cho một miền vấn đề đã chọn. Hầu hết các đặc điểm của ngôn ngữ ảnh hưởng đến khả năng đọc và khả năng ghi của chương trình. Điều này xuất phát trực tiếp từ thực tế là quá trình viết một chương trình đòi hỏi người lập trình viên phải thường xuyên đọc lại phần của chương trình đã được viết [1],[3].

Tương tự như khả năng đọc hiểu, khả năng viết cũng được đưa vào cùng một miền ứng dụng mà ngôn ngữ đó đang nhắm đến. Việc so sánh khả năng viết của hai ngôn ngữ mà miền ứng dụng mà chúng nhắm đến hoàn toàn khác nhau là bất hợp lý [1],[3].

Những mục sau đây sẽ mô tả các đặc điểm quan trọng nhất ảnh hưởng đến khả năng ghi của một ngôn ngữ lập trình.

### Tính đơn giản và tính trực giao (Simplicity and Orthogonality):

Nếu một ngôn ngữ có một số lượng lớn các cấu trúc khác nhau, một số lập trình viên có thể không quen thuộc với tất cả chúng.Việc này có thể dẫn đến việc sử dụng sai hoặc sử dụng không hiệu quả các tính năng của các cấu trúc. Vì vậy sẽ tốt hơn nếu các chương trình quy định các kiểu dữ liệu cơ bản và đề ra cho người lập trình có thể kết hợp chúng với nhau để thiết kế những giải pháp cho bài toán đặt ra thay gì có nhiều cấu trúc [1],[3].

Mặt khác, quá nhiều tính trực giao có thể gây hại cho khả năng ghi. Các lỗi trong chương trình có thể không bị phát hiện khi gần như bất kỳ sự kết hợp nào của các cấu trúc nguyên thuỷ đều là hợp lệ. Điều nà có thể dẫn đến những điểm vô lý về mã lập trình, mà trình biên dịch không thể phát hiện ra [1],[3].

### Hỗ trợ cho tính trừu tượng hoá (Support for Abstraction):

Hỗ trợ cho tính trừu tượng hóa trong lập trình ám chỉ đến khả năng sử dụng các cấu trúc phức tạp cho phép chương trình ẩn đi một số dữ liệu hoặc tiến trình. Tính trừu tượng hoá là một khái niệm chính trong lập trình đương đại. Đây là sự phản ánh vai trò trung tâm của tính trừu tượng trong các phương pháp thiết kế chương trình hiện đại. Do đó, mức độ trừu tượng hoá cho phép bởi một ngôn ngữ lập trình và tính tự nhiên của biểu thức của nó do đó rất qua trọng với khả năng ghi [1],[3].

from polygon import Polygon

class Retangle(Polygon):

def \_\_init\_\_(self, width, height):

self.width = width

self.height = height

def draw(self):

print('Retangle have width and height')

def getWidth(self):

return self.width

**Hinh 4: Lớp hình chữ nhật, cùng với các hàm lấy chiều rộng, dài, tính diện tích và chu vi**

### Tính biểu đạt (Expressivity):

Tính biểu đạt trong một ngôn ngữ lập trình là khả năng diễn đạt của ngôn ngữ đó bằng việc cung cấp các loại toán tử có tính tiện lợi cao để giúp người lập trình thuận tiện trong việc viết ra một chương trình thay vì phải sử dụng hay định nghĩa các biểu thức cấu trúc cồng kềnh [1],[3].

**Hinh 5: Các ký hiệu tăng giá trị của biến khác nhau trong ngôn ngữ**

Ví dụ, trong ngôn ngữ C ký hiệu:

Count++ // Ý nghĩa tăng giá trị của biến đếm lên 1

Thuận tiện hơn và ngắn hơn

Count = Count + 1

*Hinh : Ví dụ về tính biểu đạt trong ngôn ngữ C*

## Độ tin cậy (Reliability):

Độ tin cậy của một chương trình là khả năng chạy liên tục và không gặp bất kỳ sự xung đột nào trong quá trình chương trình hoạt động. Khả năng đọc hiểu và khả năng viết đều có sự ảnh hưởng lớn đến độ tin cậy của một chương trình [1],[3].

### Kiểm tra kiểu (Type Checking):

Kiểm tra kiểu là quá trình kiểm tra lỗi của một chương trình. Quá trình này thường được thực thi bởi trình biên dịch hoặc khi thực thi bởi trình thông dịch. Theo như quy tắc kiểm cha kiểu được thực hiện tại thời điểm biên dịch chương trình, điều này giúp tiết kiệm được thời gian tính toán, vừa giúp tìm ra lỗi của chương trình sớm hơn [1],[3].

### Xử lý ngoại lệ (Handling exception):

Khả năng giúp xử lý ngoại lệ gây cản trở chương trình và tiếp tục chạy chương trình là một khả năng tăng độ tin cậy đối với chương trình [1],[3]. Trong ngôn ngữ lập trình C++, cung cấp một cấu trúc xử lý ngoại lệ dựa trên ba từ khoá là try, catch và throw.

try

{

   // phan code duoc bao ve

}catch( ten\_Exception e1 )

{

   // day la khoi catch

}catch( ten\_Exception e2 )

{

   // day la khoi catch

}catch( ten\_Exception eN )

{

   // day la khoi catch

}

*Hinh : Ví dụ về xử lý ngoại lê trong C++*

**Hinh 6: Cấu trúc try….catch**

### Hạn chế truy cập (Restrict aliasing):

Hạn chế chuy cập được hiểu là trường hợp mà cùng lúc tồn tại hai thành phần cùng truy xuất chung một vùng nhớ. Trong trường hợp mà ngôn ngữ có hỗ trợ các khái niệm về con trỏ như C++ thì vấn đề trên sẽ rất nguy hiểm [1],[3].

**Hinh 7: Ví dụ về hạn chế truy cập**

swap\_words( uint32\_t arg ) {

uint16\_t\* const sp = (uint16\_t\*)&arg;

uint16\_t hi = sp[0];

uint16\_t lo = sp[1];

sp[1] = hi;

sp[0] = lo;

return (arg);

}

*Hinh : Ví dụ về hạn chế truy cập*

Vùng nhớ được tham trị trong *sp* sẽ là một *alias* của *arg* vì chúng cùng trỏ vào một vùng nhớ. Trong *C++*, việc tạo alias với một kiểu dữ liệu hoàn toàn khác với dữ liệu của con trỏ ban đầu. Khái niệm này được gọi là quy tắc *tên nghiêm ngặt (strict alias)*. Quy tắc này được cho phép hoạt động khi *cấp độ tối ưu hóa (optimization levels)* của trình biên dịch *gcc* trên mức *O2*. Mặc dù đoạn chương trình trên có thể được biên dịch và chạy nhưng kết quả của biến arg có thể trở thành *undefined* hoặc giá trị vẫn giữ nguyên vì quy tắc *strict alias.*

## Chi phí (Cost):

Tổng chi phí của một ngôn ngữ lập trình bao gồm các chi phí:

Đầu tiên, chi phí đào tạo các lập trình viên sử dụng ngôn ngữ, đ là một chức năng của tính đơn giản và tính trực giao của ngôn ngữ và kinh nghiệm của các lập trình viên.

Thứ hai, chi phí viết chương trình bằng ngôn ngữ. Đây là một chức năng về khả năng ghi của ngôn ngữ, điều này phụ thuộc vào tính gần gũi với mục đích của nó đối với ứng dụng cụ thể

Thứ ba, chi phí biên dịch các chương trình bằng ngôn ngữ.

Thứ tư, chi phí thực hiện các chương trình được viết bằng một ngôn ngữ bị ảnh hưởng rất nhiều bởi thiết kế của ngôn ngữ đó.

Thứ năm, chi phí của hệ thống triển khai ngôn ngữ.

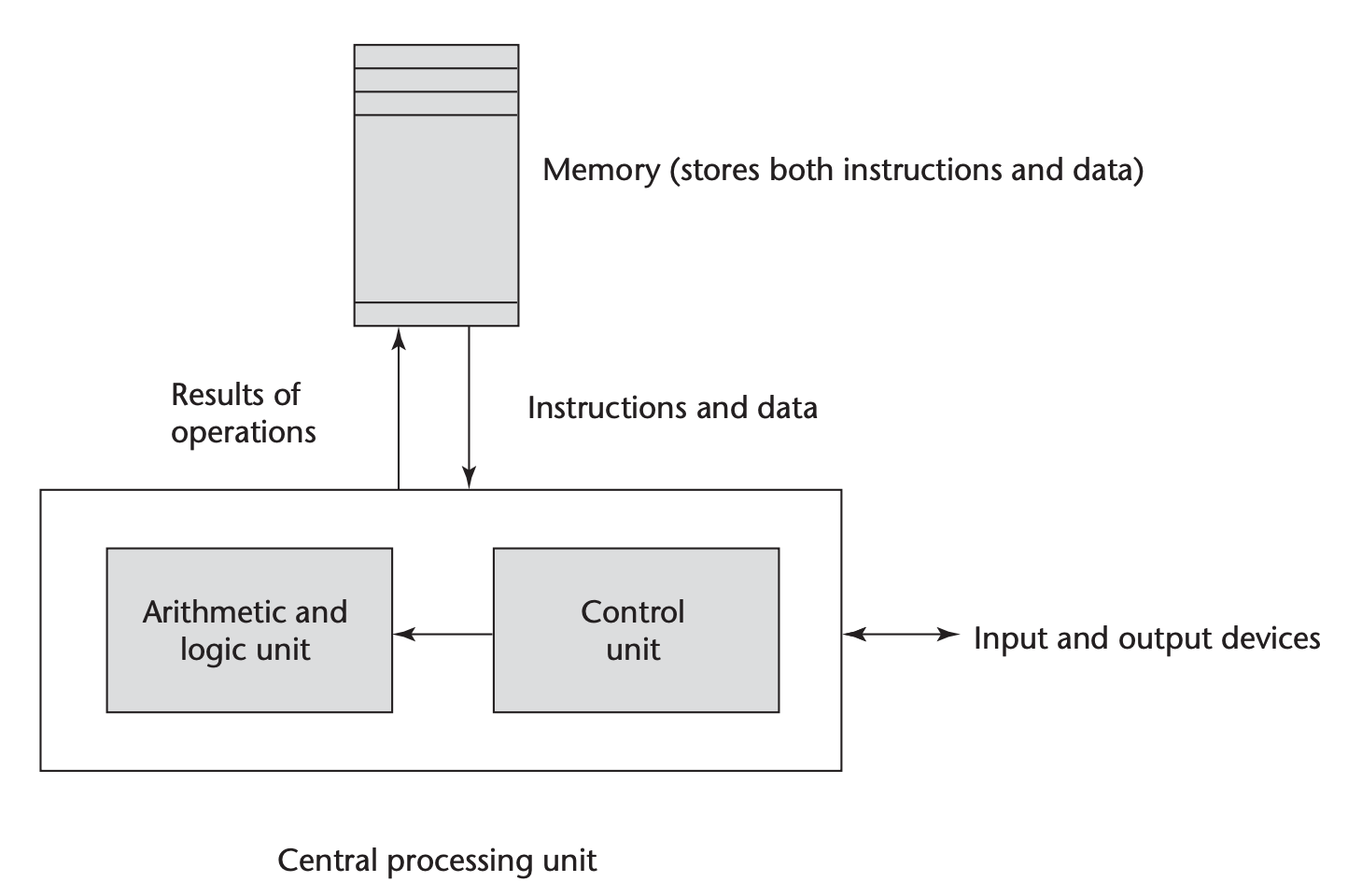
Thứ sáu, cái giá phải trả cho độ tin cậy kém và chi phí duy trì các chương trình.

# **Các yếu tố tác động, chi phối đến việc thiết kế ngôn ngữ lập trình:**

Các tiêu chuẩn để đánh giá một ngôn ngữ lập trình không chỉ là những tiêu chuẩn để người lập trình chọn lựa nên sử dụng ngôn ngữ nào mà những nhà thiết kế ngôn ngữ lập trình có thể dựa vào những yếu tố trên để thiết kế một ngôn ngữ. Ngoài các yếu tố trên thì việc một ngôn ngữ được thiết kế còn bị ảnh hưởng bởi hai yếu tố khác đó là *kiến trúc máy tính (computer architecture)* và *các* *phương pháp thiết kế lập trình (programming design methodologies)* [1],[3]*.*

## Kiến trúc máy tính:

**Hinh 8: Kiến trúc máy tính Voneuman**



*Hinh : Kiến trúc Von Neumann*

Kiến trúc *Von Neumann* là một kiến trúc máy đã định hình phương thức thiết kế của các ngôn ngữ lập trình nổi tiếng trong 50 năm qua. Khi thiết kế các ngôn ngữ mệnh lệnh, các biến được định hình dựa trên các thanh ghi, các câu gán được dựa trên *piping operation (chúng tôi tạm gọi là lệnh dẫn)*. Toán hạn trong các biểu thức được *dẫn (piped)* *từ bộ nhớ (memory)* đến CPU và các kết quả được dẫn về các *ô nhớ (memory cell)* được định nghĩa ở vế trái của biểu thức. Sự lặp lại được thực thi nhanh trong kiến trúc máy tính của *Von Neumann* vì các lệnh được lưu trữ trong các vùng nhớ liền kề nhau và quá trình lặp lại của một đoạn lệnh chỉ yêu cầu một nhánh lệnh. Sự hiệu quả này không khuyến khích cách dùng đệ quy như một phương pháp để lặp [1],[3]

Quy trình thực thi của các lệnh trong một kiến trúc *Von Neumann* được gọi là *chu kì thực thi tìm nạp (fetch-execute cycle)*.

## Các phương pháp thiết kế lập trình:

Trong những năm cuối thập niên 60 đến đầu thập niên 70, Các phong trào lập trình cấu trúc nổ ra không chỉ ở trong các lĩnh vực phát triển phần mềm mà còn ở các lĩnh vực thiết kế ngôn ngữ lập trình [1],[3].

Một trong những lý do diễn ra các phong trào lập trình cấu trúc là do sự đổi hướng về mặt chi phí tính toán từ phần cứng sang phần mềm và chi phí phần mềm được giảm nhưng chi phí về người lập trình tăng lên. Đầu những năm 60, các chương trình máy tính được làm ra chỉ đơn giản là phục vụ các vấn đề như giải các hệ phương trình hoặc để giả lập các đường bay của vệ tinh. Trong những năm tiếp theo, các nhu cầu để chương trình thay con người làm công việc phức tạp hơn tăng lên như quản lý hàng hoá, hệ thống đặt vé máy bay toàn quốc [1],[3].

Những phương pháp phát triển phần mềm mới được ra đời và dậy sóng với kết quả nghiên cứu về lĩnh vực này trong những năm 70. Các phương pháp lập trình đó được gọi là *thiết kế từ trên xuống và tinh chỉnh từng bước (top-down design and stepwise refinement)* [1],[3].

Cuối những năm 70, sự biến động từ lập trình hướng thủ tục đến lập trình hướng dữ liệu bắt đầu diễn ra. Lập trình hướng dữ liệu khởi đầu với khái niệm trừu tượng. Ngôn ngữ đầu tiên hỗ trợ tính trừu tượng có tên là *SIMULA 67,* mặc dù ngôn ngữ sau chưa được nổi tiếng với cộng đồng, nguyên nhân là do trong thời điểm đó tầm quan trọng trong việc trừu tượng hóa dữ liệu chưa được xem trọng. Tuy nhiên các ngôn ngữ sau năm những năm 70 đều được hỗ trợ tính trừu tượng hóa dữ liệu [1],[3].

Vào những năm đầu thập niên 80, lập trình hướng đối tượng được cải tiến thành lập trình hướng đối tượng. Lập trình hướng đối tượng bắt đầu với khái niệm trừu tượng hóa dữ liệu và đóng gói các tiến trình và dữ liệu thành các đối tượng. Sau đó, phương pháp lập trình hướng đối tượng bổ xung các tính chất như tính đa hình và tính thừa kế [1],[3].

# **So sánh các ngôn ngữ lập trình:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Readability** | **Writability** | **Reliability** | **Activities** |
| **C++** | Ngôn ngữ đơn giản dễ tiếp cận, dễ đọc.  Ngôn ngữ lập trình biên dịch.  Có tính trực giao. | Tính diễn đạt cao.  Có nhiều cú pháp và cấu trúc.  Số lượng thư viện hỗ trợ hạn chế.  Độ dài câu lệnh tương đối, ít hơn Java. | Nền tảng phụ thuộc.  Các hàm và các biến được sử dụng bên ngoài lớp.  Các quy tắc cú pháp được sử dụng nghiêm ngặc. | Ngôn ngữ được biên dịch thành mã máy, sử dụng trên mọi nền tảng. Thời gian xử lý nhanh. Ứng dụng Operating System, Data base, App, IoT,… |
| **Python** | Ngôn ngữ đơn giản,dễ tiếp cận, dễ đọc. Ngôn ngữ lập trình thông dịch. | Cú pháp dễ nhớ. Có hỗ trợ nhiều thư viện. Độ dài câu lệnh ngắn ít hơn JAVA và C++ 3-4 lần | Nền tảng độc lập. Các hàm và biến có thể được khai báo và sử dụng bên ngoài lớp. Các cú pháp quy tắc được sử dụng không bắc buộc. | Sử dụng trình thông dịch nên thời gian chậm hơn C++. Ứng dụng machine learning, deep learning, web Dijango,… |
| **Java** | Ngôn ngữ đơn giản, dễ tiếp cận. Ngôn ngữ lập trình biên dịch | Cú pháp dễ nhớ, tính biễu diễn cao. Độ dài câu lệnh khá lớn. | Nền tảng độc lập. Mỗi bit mã đều nằm trong cùng một lớp. Sử dụng nghiêm các cú pháp và quy tắc. | Trình biên dịch Java chậm hơn so với C++. Ứng dụng web, app android, software,… |
| **C#** | Là ngôn ngữ lập trình đối tượng cấp cao. Dễ đọc và tiếp cận. | Cú pháp đơn giản dễ trình bày. Là ngôn ngữ thông dịch. | Dành riêng cho window hiếm khi sử dụng trên các nền tảng khác. Hỗ trợ các thư viện. | Tốc độ chạy chương trình nhanh hơn Python. Ứng dụng sử dụng phát triển game đa nền tảng Unity, tạo ra nhiều ứng dụng web |
| **Pascal** | Ngôn ngữ khá đơn giản dễ đọc. Trực giao hạn chế, bởi vì có nhiều quy tắc không nhất quán | Tính biểu diễn kém. Có hỗ trợ tính trừu tượng hoá. | Các cú pháp khá đơn giản. Hỗ trợ ít thư viện. | Thời gian chạy chậm hơn C++. Ứng dụng xây dựng các ứng dụng GUI một cách nhanh chóng. |

**Phân tích các mặc và tổng hợp các kết quả việc so sánh các ngôn ngữ lập trình:**

1. **Thực hành lập trình an toàn (Secure Programming Practices):**

Theo quá trình nghiên cứu thì các ngôn ngữ như C++, Java, C#,…Là những ngôn ngữ an toàn về kiểu chữ và đặc biệt hơn hỗ trợ gõ tĩnh. JavaSript được gõ động, có nghĩa là nó kiểm tra thời gian thực thi ar các kiểu dữ liệu

1. **Phát triển ứng dụng web (Web Application Development):**

Hầu hết các ngôn ngữ cung cấp khả năng xây dựng các ứng dụng web. PHP là phổ biến nhất theo cho nghiên cứu của chúng tôi. Java xây dựng các ứng dụng web chất lượng cao bằng cách sử dụng các API nhúng và Scala có một số

các khuôn khổ web để phát triển web. Lược đồ có các cấu trúc lập trình cho phép xây dựng một máy chủ web và các ứng dụng web bằng cách mở rộng ngôn ngữ cốt lõi với macro. Haskell dựa vào bên ngoài thư viện để xây dựng các ứng dụng đó và C # có hỗ trợ rộng rãi cho việc phát triển web thông qua

Visual Studio IDE. Mặc dù BPEL có thể để lộ giao diện WSDL và các trình duyệt có thể gọi các dịch vụ do nó cung cấp, AspectJ cung cấp một cách tốt hơn để thực hiện mối quan tâm xuyên suốt mà không gây trở ngại mã hiện có. Chúng tôi nhận thấy chức năng AspectJ đã được hiển thị dưới dạng một ứng dụng web. C ++ hỗ trợ phát triển web bằng cách ẩn nhiều mạng liên lạc cùng với trình duyệt web. JavaScripts chủ yếu được sử dụng cho các ứng dụng web, bằng cách chạy một tập lệnh ở phía người dùng, giúp phân phối tính toán và giảm băng thông mạng.

1. **Thiết kế và thành phần dịch vụ web (Web Service Design and Composition):**

Tất cả các ngôn ngữ có thể thể hiện chức năng của chúng thông qua giao diện WSDL. Tuy nhiên, để soạn web dịch vụ, BPEL là ngôn ngữ đã được xây dựng cho mục đích này. C #, sử dụng .NET framework hỗ trợ rất tốt cho khả năng tương tác thông qua các dịch vụ Web. C ++ sử dụng XML-RPC giúp nhắn tin dịch vụ web và thiết lập giao tiếp trên các nền tảng khác nhau. JavaScript sử dụng API JAX-WS cho Dịch vụ web dựa trên XML để thực hiện dịch vụ web trên các nền tảng khác nhau.

1. **Tính trừu tượng hoá:**

C #, Java, Scala và PHP là các ngôn ngữ hướng đối tượng. Lược đồ là một ngôn ngữ đa mô hình và hỗ trợ mô hình hướng đối tượng. Việc triển khai chương trình của AspectJ cũng hỗ trợ một số thuộc tính của ngôn ngữ hướng đối tượng. Trong Haskell, với sự trợ giúp của các lớp kiểu, chúng ta có thể sử dụng tính năng này. C ++ là hỗ trợ phong cách lập trình dựa trên hướng đối tượng Lớp, Phiên bản, Phương thức, Truyền thông điệp, Tính đa hình Kế thừa,Trừu tượng, Đóng gói và (Kiểu con). Hỗ trợ JavaScript đơn giản mô hình dựa trên đối tượng, nó không hỗ trợ kế thừa hoặc các chức năng ảo (đa hình).

# **Kết luận:**

Các yếu tố bên trên được xem là những yếu tố chính, để giúp ta đánh giá sơ bộ được một ngôn ngữ lập trình. Bên cạnh những yếu tố trên cũng tồn tại nhiều yếu tố khác có thể coi như là một phần trong những tiêu chí để đánh giá ngôn ngữ lập trình như: *Secure Programming Practices (chúng tôi tạm gọi là các thói quen lập trình an toàn), Web Application Development (ứng dụng phát triển web), Web Service Design and Composition (chúng tôi tạm gọi là thiết kế và kết hợp dịch vụ web), OO-based Abstraction (trừu tượng hóa dựa trên hướng đối tượng), Reflection (hay còn gọi là Reflection-oriented programming chúng tôi tạm gọi là Lập trình hướng phản chiếu), Aspect-orientation (hay còn gọi là Aspect-oriented programming chúng tôi tạm gọi là lập trình hướng), Functional programming (lập trình hàm), Declarative programming (lập trình khai báo), Batch scripting, UI prototype design (thiết kế giao diện)*.

Trong quá trình nghiên cứu nhóm chúng tôi đã đưa ra các yếu tố mang tính khách quan nhất để đánh giá một ngôn ngữ lập trình. Ngôn ngữ lập trình hình thành và phát triển phải dựa trên nhiều yếu tố khác nhau, mỗi yếu tố góp phần tích cực để hoàn thiện nên một ngôn ngữ hoàn chỉnh. Tuy nhiên ta không thể hoàn toàn dựa vào những yếu tố này để kết luận là ngôn ngữ đó đã thực sự tốt hay không, hay nó còn tồn tại nhiều vấn đề. Mà chúng ta cần phải kết hợp các yếu tố cơ bản này với quá trình sử dụng và hoạt động của một ngôn ngữ để đưa ra nhận xét đúng nhất về ngôn ngữ đó.

Ngoài ra, các ngôn ngữ lập trình ngày càng phát triển không ngừng, điều này dẫn đến các yếu tố bên trên cũng sẽ thay đổi theo sự phát triển đó. Vì vậy để có một đánh giá tốt và đúng đắn về ngôn ngữ lập trình. Mối lập trình viên, người sử dụng cần phải trao dồi kiến thức, cập nhật thông tin thường xuyên để hiểu rõ bản chất của ngôn ngữ, hiểu được mục đich của nhà thiế kế. Để chọn được một hoặc nhiều ngôn ngữ tốt cho bản thân. Đồng thời góp phần hoàn thiện ngôn ngữ đó hơn.

**TÀI LIỆU THAM KHẢO**

[1] Robert W. Sebesta "Concept Of Programming language", 12th edition, Pearson Addison Wesley publisher, 2019 : 7–21.

[2] Team 10 COMP6411-S10, Comparative Studies of 10 Programming Languages within 10 Diverse Criteria revision 1.0 , pp. 94-98.

[3] https://progr-harrykar.blogspot.com/2018/11/language-evaluation-criteria