# Giáo viên hướng dẫn : Vũ Đức Vượng

# Họ tên sinh viên: Phùng Văn Hoàng

Lớp: LTU15

MSSV: 20168215

**Programming Paradigm**

Definition: Programming paradigm is a pattern that serves as a school of thought for programming of computers.

There are 4 main programming paradigms :

* **The imperative paradigm**
* **The functional paradigm**
* **The logical paradigm**
* **The object-oriented paradigm**

And others :

* The visual paradigm
* The parellel paradigm
* The concurrent paradigm
* The distributed paradigm
* The extreme paradigm

……

Some more paradigm: *message passing, procedural, value-level, function-level, flow-driven, event-driven, scalar, array, constraint, aspect-oriented, pipeline, …..*

Overview about 9 first paradigms :

# **1.The imperative paradigm.**

The “First do this and next do that” is a short phrase which really in a nutshell describes the spirit of the imperative paradigm.

Characteristics:

* Discripline & idea: Digital harwear and the ideas of Von Neumann, features closely related to machine architecture
* Use statements that directly changes a program’s state (datafields)
* Focus on describing how the program operates
* Contrast to declarative paradigm
* Straightfoward abstractions of the way a tradditional Von Neumann computer works
* Language representatives: Fortran, Algol, Pascal, C, Basic, Python, Ruby,…

# **2. Functional paradigm**

Functional programming is much simpler and more clean than imperative programming because it originates from mathematical discripline: The theory of functions.

Characteristics :

* Discription & Idea : Mathametic and theory of functions
* Abtracts a single expression to a function which can be evaluate
* Avoid side effect : changing state and mutable data
* The result depends only on the arguments, doesn’t depend on the local or global states.
* Common in academia than commercial software development
* Fit well with computations driven by needs
* Language representatives : C++, Clojure, F#, Haskell, Lisp, Python, Ruby, JavaScript, …

# **3. Logical paradigm**

Characteristics:

* Discripline & idea: largely base on formal logic
* A logical program is a collection of axioms expressing the facts and rules about some problem domain
* User states a goal to be proven
* The logical programming system uses reference steps to prove the problem
* Language representatives : Prolog, Datalog, ASP.

# **4. Object – Oriented paradigm**

Send messages between objects to simulate the temporal evolution of a set of real world phenomena

Characteristics :

* Discripline & Idea : The theory of concept, models of human intereaction with real world phenomena
* Data as well as operations are encapsulted in objects
* Information hiding is used to protected internal properties of an object
* Objects are grouped in classes: classes represent concept whereas objects represent phenomena
* Language representatives : Java, C++, C#, Python, PHP, Ruby, Perl, Swift, Objective C,…

Other programming paradigm:

# **5. Visual paradigm**

The combination of Graphic, Programming language and human-computer intereaction.

Characteristics:

* Discripline & Idea : The poinner : Sketchpad System of Ivan Sutherland (1963), the first graphics application on computer.
* User creat program by manipulating the program elements graphically rather than specifying them textually.
* Build program in Event-Driven Programming
* Programer creat interface throughout operations on screen that intereact objects (button, view, box, list,..) and properties (color, size, style, …) with few text code.
* IDE read the module that programer design then compile it for operation by computer.

# **6. Parallel paradigm**

Process mutil tasks at the same time

Characteristics:

* An abtraction of parallel computer architecture.
* Shared memory
* Message passing
* Multiple threads
* Data parallel
* Language representatives: C, C++, ..
* Parallel libraries: MPI, OpenMP, OpenCL, CUDA, …

# **7. Concurrent paradigm**

Operations are executed during overlapping time periods concurrently.

* The paralle programming is not possible on One – core processor but this is.
* Consist of process lifetime overlapping
* Execution don’t happen at the same instant
* Laguage representatives : Java, C#, C++, JavaScript, NodeJs, Erlang, Ada, Julia, …

# **8. Distributed paradigm**

A distributed program mudel is an abtraction layer above comunication protocols details.

Discripline & idea : Distributed Algorithm.

Characteristics :

* Each processor has its own private memory ([distributed memory](https://en.wikipedia.org/wiki/Distributed_memory))
* Information is exchanged by passing messages between the processors.
* Architectures: Client - server, three - tier, n- tier, Peer – to – peer, …
* Application : telecommunication network, network applications, realtime process control, parallel computation,….

# **9. Extreme paradigm**

Vietnamese name: Lập trình GẮT

Author : Ken Beck

Characteristics:

* Is a software development methodology, a type of agile software development
* Improve software quality and responsiveness to changing customer requirements
* organizes people to produce higher-quality software more productively
* Activities : coding, testing, listening, and designing
* Values : communication, simplicity, feedback, courage and respect.

# **Visual paradigm programming:**

*(Vì trình độ tiếng anh còn kém, nên xin được viết phần này bằng ngôn ngữ Tiếng Việt*)

Visual paradigm programming hay còn gọi là mô hình lập trình trực quan, là một mô hình lập trình mới, có tính ứng dụng cao, tính phổ biến cao và là một phần trong quá trình phát triển của các hình thức lập trình.  
Tóm lược về quá trình tiến hóa của các hình thức lập trình:

## **1.Hình thức lập trình nguyên thủyx: Mã máy**

Thời kì đầu (First – Generation programming language), chưa có ngôn ngữ lập trình hiện đại nào, thì phải lập trình trực tiếp bằng mã máy. Mã byte được đục lỗ lên các phiếu đục lỗ (punched card), sau đó cho đọc bởi máy đọc phiếu rồi tới máy tính. Về sau, chương trình được lưu bởi đĩa từ để tốc độ đọc nhanh hơn nhưng về hình thức vẫn là chương trình biểu diễn bởi dạng mã máy nguyên thủy.

Việc lập trình lúc này phục thuộc nhiều vào phần cứng, mã máy chưa thống nhất giữa các hãng sản xuất phần cứng khác nhau. Công việc lập trình lúc này chỉ dành cho các chuyên gia về khoa học máy tính, hiểu được nguyên lý hoạt động sâu của chương trình viết bằng mã máy và cách sử dụng mã máy, cùng với hoạt động phần cứng.

## **2.Hình thức lập trình: Text**

* **Ngôn ngữ lập trình lai giữa ngôn ngữ thời kì đầu (First- generation programming language) và ngôn ngữ tự nhiên của con người: Hợp ngữ**

Ở giai đoạn này, mã lệnh được thay bằng các tên gợi nhớ, chương trình được viết dưới dạng text sau đó được dịch sang mã máy. Ngôn ngữ lập trình thời kì này được coi là thế hệ thứ 2(Second-generation programming language). Lập trình thuận lợi hơn, khi dịch có thể liene kết thư viện. Tuy vậy người lập trình vẫn phải nắm chắc về hoạt động của hệ thống xử lý, về các thanh ghi, con trỏ và vai trò của chúng thì mới tránh được lỗi.

Lúc này bắt đầu có sự thống nhất quốc tế về các tập lệnh cần thiết. Số lượng người có thể lập trình đã rộng hơn thời kì sử dụng mã máy nhưng vẫn còn khó khăn do sự cản trở của các cú pháp lập trình ít ý nghĩa.

* **Ngôn ngữ lập trình hình thức, gần với ngôn ngữ tự nhiên : Ngôn ngữ bậc cao**

Ngôn ngữ lập trình bậc cao hay ngôn ngữ lập trình thế hệ thức 3 (Third – generation programming language) ra đời vào những năm 1950, ít lệ thuộc phần cứng. Các đại diện thời kì đầu của giai đoạn : Fortran, COBOL, BASIC, Pascal, C, … Hiện nay có : C, C++, Java, JavaScrips, Python, PHP, Ruby, … Ngày nay các ngôn ngữ lập trình đều đi kèm với môi trường soạn thảo hỗ trợ gợi ý, kiểm tra cú pháp, gỡ lỗi,…

Lúc này sự phổ biến của công việc lập trình rất phổ biến, các ngôn ngữ lập trình đều gần gũi với ngôn ngữ tự nhiên, tường minh, sự hỗ trợ của IDE cũng như các thư viện khiến việc lập trình dễ dàng, ít phải nhớ máy móc cú pháp thiếu nghĩa, lập trình một ngôn ngữ chỉ phục thuộc hệ điều hành giúp lập trình viên sử dụng các ngôn ngữ một cách linh hoạt trên nhiều nền tảng phần cứng. Tuy vậy, việc lập trình vẫn khó khăn đối với những người không có chuyên môn về ngành.

## **3.Hình thức lập trình lai giữa Text và Giao diện đồ họa (Visual programming hiện nay)**

Lập trình viên lập trình bằng việc kéo thả, kết nỗi, lựa chọn các option của các đối tượng mà trình thiết kế cung cấp để tạo một chương trình hoàn chỉnh. Các đối tượng thành phần đa dạng được biểu diễn một cách trực quan bằng đồ họa hình ảnh, mỗi tương quan của các thành phần được biểu diễn một cách tường minh, tổng quát qua bố cục mà trình thiết kế cung cấp giúp lập trình viên dễ dàng nhận biết và tương các với các thành phần đó, thiết lập các thuộc tính cho các đối tượng. Tuy vậy, những phần mềm giúp thiết kế chương trình hiện nay là chưa Đồ họa hóa chương trình một cách triệt để, khi phải tương tác sâu vào đối tượng, vẫn phải sử dụng dạng lập trình text để giải quyết vấn đề.

Đối với hình thức lập trình này, chương trình được biểu diễn 1 cách khoa học, sáng nghĩa thông qua đồ họa, giúp việc lập trình không chỉ giành cho người có chuyên môn mà cho cả những đối tượng thiếu chuyên môn nhưng đủ khả năng logic và thiết kế logic cho một chương trình. Các dòng lệnh được minh họa bằng hình ảnh có nghĩa điện diện khiến việc nhớ cú pháp không phải là vấn đề.

## **4.Hình thức lập trình mô phỏng kịch bản (Tương lai)**

* Vấn đề gì ở hình thức mô tả chương trình?

1. Sự bất lực của ngôn ngữ tự nhiên.

“Làm sao ta có thể giải thích cho ai đó biết một trò chơi là gì? Tôi cho rằng ta sẽ mô tả cho anh ta những trò chơi và có thể bổ sung vào việc mô tả ấy câu nói: “đó, và những thứ giống như thế, người ta gọi là “trò chơi”. Và phải chăng bản thân ta biết nhiều hơn? Phải chăng chỉ vì ta không thể nói chính xác cho người khác biết một trò chơi là gì? – Nhưng, đây không phải là việc không biết. [Bởi] quả thật ta không biết những ranh giới, vì không có ranh giới nào được vạch ra hết”.Trích mục 1 bài báo :”Trò chơi ngôn ngữ “trong Wittgenstein và luận đề “Ngôn ngữ là ngôi nhà của tồn tại” của Martin Heidegger”.

Ngôn ngữ là một cách thức giao lưu tuyệt vời, nhưng không phải là một phương pháp tối ưu để mô tả chân lý. Ngoài ví dụ đề cập ở trên thì một minh chứng khác thú vị không kém cho sự “yểu” của ngôn ngữ hiện nay hoặc chữ viết: ngôn ngữ hiện nay cả dạng nói và viết đều là tuyến tính, ta viết và dịch câu chữ theo một chiều xác định(trên xuống hoặc trái phải), ta cũng hiểu nội dung nói theo thứ tự xuất hiện của thông tin theo trục thời gian(trước sau), vì thế chúng thích hợp biểu diễn những mô hình tuyến tính, nhưng lại làm dở tệ đối với mô hình phi tuyến tính, bằng chứng là việc để hiểu những mô hình phi tuyến tính, ta cần nhiều công sức hơn và cảm thấy khó hơn nhưng thực ra cá nhân tôi nghĩ đó là một triệu trứng của việc đem ngôn ngữ biểu diễn tuyến tính để đi biểu diễn một mô hình phi tuyến tính.

Việc lập trình có thể nói là quá trình mô tả một sự vận hành chính xác(gần giống như chân lý), nó không nên được thực hiện thông qua chữ viết – một hình thái khác của ngôn ngữ, vốn dùng để giao lưu và bao hàm trong chính nó một lớp nghĩa phụ thuộc vào hoàn cảnh (không nhất quán). Đó là đặc điểm của ngôn ngữ người với người, còn về ngôn ngữ người với máy (ngôn ngữ lập trình) do được thể hiện thông qua hình thức chữ viết có cấu trúc tương tự ngôn ngữ tự nhiên và dựa vào ngôn ngữ tự nhiên nên nó cũng mang các đặc điểm khó chịu tương tự . Vốn dĩ ngôn ngữ tự nhiên đã là một trở ngại cho việc mô tả chân lý, phù hợp với việc biểu diễn các thông tin mang tính khả biến, tùy biến vào hoàn cảnh, thiếu nhất quán, thứ mà một ngôn ngữ dùng để mô tả một chương trình máy tính hay một quy trình rất kiêng ngại bởi những mô tả như thế chú trọng nhất vào tính chính xác tuyệt đối. Vì thế, việc lập trình nói riêng và mô tả các quy trình, ý tưởng nói chung nên được thực hiện thông qua một hình thức mới trực quan hơn, phổ quát hơn, một hình thức mà có thể mục đích chỉ để cho việc lập trình nhưng có thể là tiền đề cho việc thay đổi cách biểu đạt thông tin, giao tiếp của con người.

1. Sự phình cấu trúc mô tả chương trình khi dùng ngôn ngữ tự nhiên.

Chữ viết là một dạng biểu diễn thông tin. Hiện tại, ở mọi ngôn ngữ, số lượng chữ cái trong bảng chữ cái đều có hạn, các từ đơn lẻ được tạo từ việc ghép các chữ cái đơn lẻ. Theo sự phát triển của công nghệ thông tin, khoa học máy tính và các ngành liên đới với khoa học máy tính, lượng kiến thức mới sẽ theo đó tăng lên 1 cách rất nhanh, yêu cầu đặt ra là phải tạo ra những khả năng biểu diễn mới đáp ứng việc biểu diễn 1 cách rõ ràng các kiến thức mới này. Có 3 cách tạo ra các khả năng biểu diễn mới trong ngôn ngữ hiện này : (i) Tạo chữ cái mới (ii) Tạo ra những từ mới (iii) Tạo ra những cụm từ mới bằng việc kết hợp những từ đã có sẵn. Phương pháp tạo cụm từ thứ (iii) được sử dụng nhiều nhất hiện nay. Ta có thể thấy các ví dụ điển hình: object – oriented programming, human – computer intereaction, intergrated development enviroment, … Theo thời gian, để mô tả cho những kiến thức phức tạp và liên đới ngành hơn, cụm từ mới được sáng tạo ra sẽ có cấu trúc phức tạp hơn và kích cơ lớn hơn. Điều này là một cản trở đối với việc trao đổi và biểu diễn thông tin.

Điều này xảy ra tương tự đối với các ngôn ngữ lập trình dạng văn bản hiện nay. Để đáp ứng nhu cầu viết ra những chương trình có cấu trúc phức tạp hơn, quy mô lớn hơn, chúng ta đơn giản chỉ là viết code nhiều hơn, cố gắng quản lý mã nguồn logic hơn, chia nhỏ các tính năng để thiết kế riêng biệt. Điều này có vẻ vẫn có hiệu quả cho tới ngày nay, nhưng chắc chắn đang trở thành một vấn đề của tương lai khi mà quy mô chương trình vượt khả năng quản lý của con người, cũng như cách tư duy lập trình theo lối ngôn ngữ truyền thống khiến quá trình thiết kế hệ thống, phần mềm chậm chạp, thiếu khả năng mô phỏng những quá trình vận hành, mô hình phức tạp, trừu tượng.

* Một ý tưởng nhỏ cho việc thay đổi.

Một giải pháp tham vọng cho hình thức lập trình tương lai, gọn hơn, rõ hơn, tường minh hơn, dễ dàng hơn, bao quát hơn và gần với bản chất của tư duy hơn đó là Lập trình mô phỏng kịch bản.

Tư duy con người về các quy trình như cách mà một chương trình làm việc hoặc về một mô hình, hệ thống,… chưa bao giờ được biểu diễn ở dạng chữ viết khi nó còn nằm ở trong đầu người lập trình. Nhu cầu trao đổi thông tin và lưu trữ ý tưởng, kiến thức khiến con người phải biểu diển ý tưởng theo dạng chữ viết, vì thế cũng phải lập trình dạng chữ viết. Việc tạo ra một phương pháp trao đổi thông tin mới gần với cách tư duy “mô phỏng kịch bản “ của não bộ con người sẽ giải quyết được vấn đề độ phình của lực lượng chữ viết cũng như tính phức tạp, rời rạc của một chương trình máy tính viết bằng chữ viết, cung cấp cho lập trình viên những cái nhìn tổng quan hơn về dự án, chương trình, mô phỏng thực tế cách chương trình hoạt động, gần gũi và đơn giản hơn.

Vậy vấn đề đặt ra là phải xây dựng một hệ thống trao đổi và biểu diễn thông tin mới mà không dựa trên nền tảng chữ viết. Tôi nghĩ cho đến giờ, “kịch bản “ vẫn là một hình thức biểu diễn tiềm năng, mặc dù chắc chắn tồn tại những hình thức biểu diễn khác thực sự tốt hơn và gần với dạng “raw” của thông tin mà não bộ lưu trữ hơn. Chúng ta vẫn đang phát triển các hình thức lập trình trực quan thiên hướng đồ họa hơn và dần loại bỏ text. Quan hệ giữa các đối tượng trong kịch bản sẽ được mô phỏng một cách chính xác như trong nhận thức của người lập trình. Ưu điểm của việc sử dụng đồ họa là nó khá gần với dạng tồn tại của kiến thức trong nộ não người đó là những kí ức dựa trên sự kiện được biểu diễn bằng hình ảnh kết hợp âm thanh, mùi vị, cảm xúc, và hơn nữa đồ họa, âm thanh, … là những dạng biểu diễn phi tuyến tính, nó giải quyết được khó khăn của chữ viết (âm thanh ở đây khác với tiếng nói đề cập phía trên về sự tuyến tính, tiếng nói là các chuỗi âm thanh vì thế nó tuyến tính, âm thanh riêng rẽ thì không).

Tuy nhiên giải pháp vẫn còn có thể tiến xa hơn nữa. Thứ nhất, ta có thể hủy bước trung gian của việc biểu diễn thông tin, thay vì lập trình kéo thả đồ họa trên máy tính, hay tương tác trực tiếp với não bộ. Ta đâu cần tay để mô phỏng một kịch bản trong suy nghĩ Thứ hai, ta có thể tiếp cận gần hơn tới dạng “raw” của thông tin được biểu diễn trong não bộ. Một khi làm được điều này, việc tạo ra một kịch bản sẽ rất trực tiếp và việc trao đổi các kịch bản này giữa người với người sẽ thuận lợi hơn bởi vì bản chất là trao đổi kí ức cho nhau. Nghe có vẻ hoang đường, nhưng về lý thuyết là có thể, nó có tiềm năng ứng dụng và chắc chắn chúng ta sẽ đạt được những điều này trong hành trình tìm hiểu bộ não của chính mình. Tôi cá là sẽ chẳng ai trong lứa bọn tôi sẽ sống đủ lâu để nhìn thấy sự biến chuyển này, hoặc thậm chí họ sẽ chẳng tin, hoặc họ chẳng đủ sự quan tâm hay kiến thức để tin và hiểu.

Tóm lại, một ý tưởng cho tương lai, các lập trình viên sẽ chỉ cần tưởng tượng để hoàn thành một chương trình và việc nhớ cú pháp lập trình, gõ bàn phím như tôi đang làm sẽ là một phần lịch sử của quá trình phát triển của các hình thức lập trình trực quan.

# **Tham khảo:**

<http://people.cs.aau.dk/~normark/prog3-03/pdf/paradigms.pdf>

<https://en.wikipedia.org/wiki/Comparison_of_programming_paradigms>

<https://www.wikipedia.org/>