**SOMETHING ABOUT OUR PROJECT**

**Read me:**

***“ Phần 1: Những điều cần biết, có lẽ biết rồi***

***Phần 2: Phân tích các giai đoạn khi viết phần mềm***

***Phần 3: Các phương pháp xây dựng phần mềm bài bản(nhiều mà đọc chán m có skip)***

***Phần 4: Phần mềm nên được viết trên nền tảng nào – Đọc cái này nè 🡸 ”***

1. **CẦN BIẾT**

* Nhân sự: Ai sẽ làm? Ai làm gì?
* Thời gian: Khi nào làm? Làm mất bao nhiêu thời gian?
* Phương pháp: Làm như thế nào?
* Công cụ: Dùng công cụ gì để làm công việc này?
* Chi phí: Chi phí bỏ ra bao nhiêu? Thu về bao nhiêu? (ước tính)
* Mục tiêu: Mục tiêu hướng đến là gì?

Có 4 thao tác là nền tảng của hầu hết các quy trình phát triển phần mềm:

* Đặc tả phần mềm: Định nghĩa được các chức năng, điều kiện hoạt động của phần mềm.
* Phát triển phần mềm: Là quá trình xây dựng các đặc tả.
* Đánh giá phần mềm: Phầm mềm phải được đánh giá để chắc chắn rằng ít nhất có thể thực hiện những gì mà tài liệu đặc tả yêu cầu.
* Tiến hóa phần mềm: Đây là quá trình hoàn thiện các chức năng cũng như giao diện để ngày càng hoàn thiện phần mềm cũng như các yêu cầu đưa ra từ phía khách hàng.

To be continued….

1. **Các giai đoạn**

Có 5 giai đoạn chính trong qui trình phát triển phần mềm: Phân tích (**Analysis**), Thiết kế (**Design**), Cài đặt mã nguồn (**Coding**), Kiểm thử (**Testing**) và Vận hành (**Operation**). Phần mềm ra đời dựa trên yêu cầu của người dùng.

**PHÂN TÍCH**

[](https://3.bp.blogspot.com/-bzrRimDDq5Y/VxfPSW7J6wI/AAAAAAAAAcM/C4T-QiAhmL88CUv3ap7LxzwWZaTlIoqsQCLcB/s1600/231c098.jpg)

Ví dụ: một người muốn có một danh bạ online. Trong giai đoạn phân tích, chúng ta thực hiện nghiên cứu tính khả thi, chúng ta phân tích các vấn đề và xác định xem liệu giải pháp có thực sự khả thi. Và nếu giải pháp đó khả thi, thì kết quả của giai đoạn này sẽ là bảng "***Đặc tả yêu cầu****"  - requirement specification -*để mô tả các chức năng của chương trình. Các tính năng này phải được phát biểu theo những cách có thể kiểm chứng được. Một trong những tính năng của danh bạ online là có khả năng tìm kiếm một người dựa trên first name của họ. Chúng ta có thể kiểm tra tính năng đó bằng cách chạy trực tiếp chương trình và tìm kiếm, và kiểm tra xem chương trình có hoạt đông đúng và cụ thể khi tên của một người có trong danh bạ, và người không có trong danh bạ được gõ vào khung tìm kiếm. Những việc làm đó thuộc pha *Kiểm thử*, thứ mà mình sẽ nói tới nó sau.

**THIẾT KẾ**

Trong giai đoạn thiết kế*,*chúng ta sẽ chuyển bảng đặc tả yêu cầu thành bảng "***Thiết kế chi tiết****" - detailed design*của chương trình.   
Trong giai đoạn này chúng ta sẽ**thiết kế các giao diện** người dùng của chương trình (User Interface) -  bao gồm các bước: Lập danh sách các màn hình (view, form,...) dựa vào Use Case, vẽ sơ đồ mối quan hệ giữa các màn hình, Thiết kế các đối tượng trên mỗi màn hình (Sắp xếp vị trí các button, textbox,...) theo các qui tắc về thiết kế giao diện, cuối cùng đặc tả, giải tích cách hoạt động của các đối tượng trên từng màn hình đó.  
Với một chương trình thiết kế hướng đối tượng (object-oriented design), kết quả của pha này không thể thiếu Danh sách các lớp (class) được dùng để đáp ứng yêu cầu. Chúng ta sẽ **thiết kế các lớp đối tượng**, dựa vào những yêu cầu cần thiết, các kiến thức về kế thừa, đa hình để thiết kế các lớp đối tượng, và mô hình hóa chung bằng các sơ đồ (ví dụ UML - Unified Modeling Language). Ví dụ: với chương trình danh bạ online ở trên thì chúng ta có thể cần các class như Person, Phone, Group,...

|  |
| --- |
| [https://3.bp.blogspot.com/-jZ0XToIyI68/VxfPddIQ2BI/AAAAAAAAAcQ/Kdz0bhRqHwkzEp8wbOpFFouDSIr68TAiQCLcB/s640/software-PM-services.png](https://3.bp.blogspot.com/-jZ0XToIyI68/VxfPddIQ2BI/AAAAAAAAAcQ/Kdz0bhRqHwkzEp8wbOpFFouDSIr68TAiQCLcB/s1600/software-PM-services.png) |
| Thiết kế phần mềm - giai đoạn không thể thiếu trong qui trình phần mềm |

Ngoài ra ở một số chương trình đặc thù, ta còn cần **thiết kế các tầng dữ liệu** cho chương trình. Dữ liệu ở đây có thể là hệ thông tập tin đơn giản, đến các hệ  cơ sở dữ liệu phức tạp. Sau đó vẽ sơ đồ mô hình hóa, đặc tả dữ liệu.  
Cuối cùng không thể thiếu là **thiết kế xử lý** cho chương trình. Chi tiết hóa các kịch bản usecase, các luồng logic trong chương trình. Và sử dụng sơ đồ tuần tự để mô tả.

**CÀI ĐẶT MÃ NGUỒN**

Trong giai đoạn **cài đặt mã nguồn**, chúng ta triển khai các thiết kế thành một chương trình thực tế bằng các ngôn ngữ lập trình như Java, C#, Python, hay các hàm API,... Chúng ta đã có một bảng thiết kế cấu trúc hoàn chỉnh, thì việc triển khai thành mã nguồn không thực sự quá khó khăn.

|  |
| --- |
| [https://2.bp.blogspot.com/-XoNozj9Yl_A/VxfP4YznOkI/AAAAAAAAAcY/k3tdVHwFm6gA4SkUSwv6wSa204tM5BQ2gCLcB/s640/photo-1429051883746-afd9d56fbdaf.jpg](https://2.bp.blogspot.com/-XoNozj9Yl_A/VxfP4YznOkI/AAAAAAAAAcY/k3tdVHwFm6gA4SkUSwv6wSa204tM5BQ2gCLcB/s1600/photo-1429051883746-afd9d56fbdaf.jpg) |
| Nhiều người hay lầm tưởng làm phần mềm chỉ là ngồi viết code! |

**KIỂM THỬ**

Khi quá trình triển khai mã nguồn hoàn chỉnh, chúng ta sẽ đến với giai đoạn kiểm thử. Trong giai đoạn này chúng ta sẽ chạy chương trình với nhiều bộ dữ liệu để kiểm chứng là chương trình chạy đúng theo đặc tả yêu cầu. Hai loại kiểm thử dành cho các chương trình hướng đối tượng là: kiểm thử đơn vị (**unit testing**) và kiểm thử tích hợp **(integration testing**).

|  |
| --- |
| [https://4.bp.blogspot.com/-BkA0P4iWjQM/VxfP3Tso-II/AAAAAAAAAcU/ndCSZyU9PVIYOWLON_sI4uW-MJrYiPpuwCKgB/s640/software-Testing.png](https://4.bp.blogspot.com/-BkA0P4iWjQM/VxfP3Tso-II/AAAAAAAAAcU/ndCSZyU9PVIYOWLON_sI4uW-MJrYiPpuwCKgB/s1600/software-Testing.png) |
| Giai đoạn dễ gây "mâu thuẫn" giữa các thành viên trong team nhất! |

Kiểm thử đơn vị thường do lập trình viên thực hiện, kiểm nghiệm từng class riêng biệt, từng hàm trong mã nguồn trong môi trường cô lập. Còn đối với kiểm thử tích hợp chúng ta kiểm tra các class có làm việc đúng khi ghép lại với nhau hay không, và quá trình test diễn ra ngay sau unit testing. Hành động "bất hủ" dùng để phát hiện và loại bỏ lỗi của quá trình  thiết kế và cài đặt gọi là "**debugging**". Nếu tìm được lỗi, chúng ta phải quay về pha trước đó để sửa chữa và hoàn thiện chương trình.  
  
Bonus cho các bạn 1 câu nói bất hủ của **Dijkstra**:

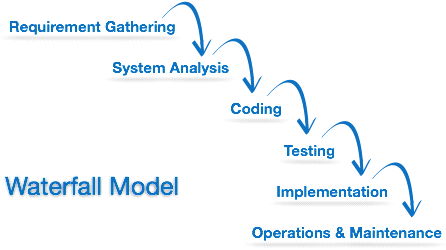
*"Program testing can be used to show the presence of bugs, but never to show their absence!"*

**VẬN HÀNH**

Cuối cùng sau khi quá trình kiểm thử kết thúc thành công, chúng ta đi vào pha vận hành, khi đó chương trình sẽ được đưa vào sử dụng thực tế. Thứ quan trọng nhất và mất nhiều thời gian nhất trong pha này là **bảo trì phần mềm** (*software maintenace*). Ngay cả sau khi phần mềm được đưa vào sử dụng, chúng ta hầu như luôn phải sửa đổi nó. Bởi vì khách hàng có thể yêu cầu thêm tính năng, hoặc các lỗi mới được tìm thấy.   
Thống kê cho thấy, xấp xỉ 70% phí của phần mềm thuộc về công đoạn bảo trì. Vậy nên khi phát một phần mềm chúng ta phải nhắm vào phần mềm dễ bảo dưỡng, bỏ thời gian và công sức ra để phân tích thiết kế và lập trình cẩn thận. Ngay cả nó có mất thời gian, và chi phí giai đoạn đầu, nhưng trong qua trình hoạt động lâu dài, các phần mềm có sự chuẩn bị thiết kế chu đáo sẽ ít tốn kém hơn. Đây là một điểm rất quan trọng mà các bạn nào muốn trở thành một nhà phát triển phần mềm giỏi cần phải lưu ý!  
  
--------------------------------------------------------------------------------------------

|  |
| --- |
| [https://4.bp.blogspot.com/--ktuOUBEsSI/VxfMz9KCQEI/AAAAAAAAAb0/yyaFn-o6xCQP0lwNtUUgdPZmLZLnc7rdgCLcB/s640/Helpful-Tips1%2B%25281%2529.jpg](https://4.bp.blogspot.com/--ktuOUBEsSI/VxfMz9KCQEI/AAAAAAAAAb0/yyaFn-o6xCQP0lwNtUUgdPZmLZLnc7rdgCLcB/s1600/Helpful-Tips1%2B%25281%2529.jpg) |

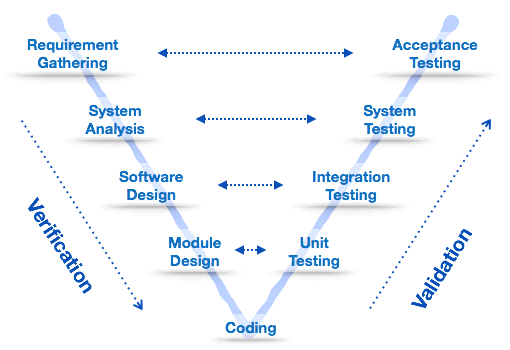
1. **Các mô hình phát triển phần mềm**
2. **Waterfall model- Mô hình thác nước**



* + Mô tả:
    - Mô hình thác nước là mô hình áp dụng theo tính tuần tự của các giai đoạn phát triển phần mềm
    - Có nghĩa là: giai đoạn sau chỉ được thực hiện tiếp khi giai đoạn trước đã kết thúc
    - Không được quay lại giai đoạn trước để xử lí các thay đổi trong yêu cầu
    - Đây được coi là mô hình phát triển phần mềm đầu tiên.
  + Áp dụng:
    - Thường được áp dụng cho các dự án không thường xuyên bị thay đổi về yêu cầu.
  + Đặc điểm
    - Ưu điểm:
    - Dễ sử dụng, dễ tiếp cận
    - Các giai đoạn và hoạt động được xác định rõ ràng
    - Xác nhận ở từng giai đoạn, đảm bảo phát hiện sớm các lỗi
    - Nhược điểm:
    - Rất khó để quay lại giai đoạn nào khi nó đã kết thúc
    - Ít tính linh hoạt và phạm vi điều chỉnh của nó khá là khó khăn, tốn kém.

Tham khảo thêm: <https://melsatar.wordpress.com/2012/03/15/software-development-life-cycle-models-and-methodologies/>

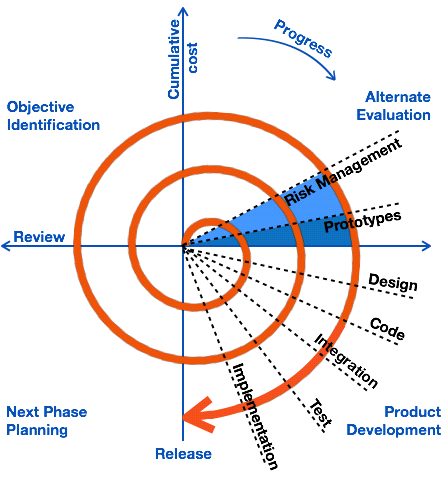
1. **V- Shaped Model- Mô hình chữ V**



* Mô tả:
* Đây là mô hình mở rộng từ mô hình thác nước
* Thay vì di chuyển xuống theo tuần tự các bước thì quy trình sẽ đi theo hình chữ V
* Áp dụng:
* Yêu cầu phần mềm phải xác định rõ ràng
* Công nghệ phần mềm và các công cụ phải được tìm hiểu kĩ
* Đặc điểm:
* Ưu điểm:
  + Đơn giản dễ sử dụng
  + Phấn phối cụ thể theo mỗi giai đoạn
  + Thực hiện verification và validation sớm trong mỗi giai đoạn phát triển
* Nhược điểm:
  + Phạm vi điều chỉnh khá là khó khăn và tốn kém.

Tham khảo thêm: <https://melsatar.wordpress.com/2012/03/15/software-development-life-cycle-models-and-methodologies/>

**3. Spiral Model – Mô hình xoắn ốc**



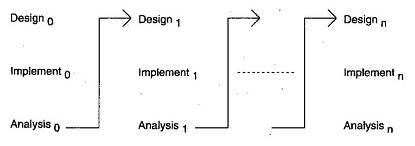
* Mô tả:
  + Là mô hình kết hợp giữa các tính năng của mô hình prototyping và mô hình thác nước.
  + Mô hình xoắn ốc được ưa chuộng cho các dự án lớn, đắt tiền và phức tạp.
  + Mô hình này sử dụng nhiều những giai đoạn tương tự như mô hình thác nước, về thứ tự, plan, đánh giá rủi ro, …
* Áp dụng:
  + Thường được sử dụng cho các ứng dụng lớn và các hệ thống được xây dựng theo các giai đoạn nhỏ hoặc theo các phân đoạn
* Đặc điểm:
  + Ưu điểm:
    - Estimates (i.e. budget, schedule, etc.) trở nên thực tế hơn như là một quy trình làm việc, bởi vì những vấn đề quan trọng đã được phát hiện sớm hơn.
    - Có sự tham gia sớm của deverlopers
    - Quản lý rủi ro và phát triển hệ thống theo phase
  + Nhược điểm:
    - Chi phí cao và thời gian dài để có sản phẩm cuối cùng
    - Phải có kỹ năng tốt để đánh giá rủi ro và giả định.

Tham khảo thêm: <https://melsatar.wordpress.com/2012/03/15/software-development-life-cycle-models-and-methodologies/>

**4. Iterative Model- Mô hình tiếp cận lặp**

Example: 

Diagram:



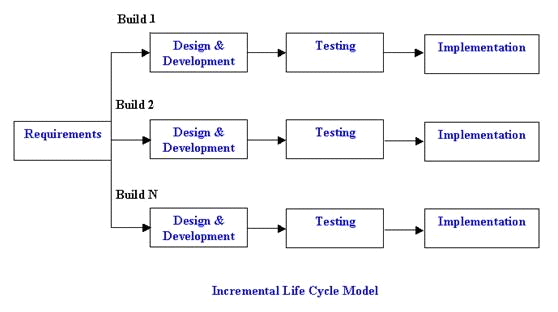
* Mô tả:
  + Một mô hình được lặp đi lặp lại từ khi start cho đến khi làm đầy đủ spec
  + Thay vì phát triển phần mềm từ spec đặc tả rồi mới bắt đầu thực thi thì mô hình này có thể review dần dần để đi đến yêu cầu cuối cùng.
  + Quy trình phát triển được lặp đi lặp lại cho mỗi một version của sản phẩm trong mỗi chu kỳ.
  + Áp dụng:
  + Yêu cầu của hề thống đã hoàn chỉnh, được xác định rõ ràng và dễ hiểu
  + Yêu cầu chính cần được xác định, và một số chi tiết có thể được đổi mới theo thời gian
  + Đặc điểm
  + Ưu điểm:
    - Xây dựng và hoàn thiện các bước sản phẩm theo từng bước
    - Nhận được phản hồi của người sử dụng từ những bản phác thảo
    - Thời gian làm tài liệu sẽ ít hơn so với thời gian thiết kế
  + Nhược điểm:
    - Mỗi giai đoạn lặp lại thì cứng nhắc
    - Tốn kiến trúc hệ thống hoặc thiết kế các vấn đề có thể phát sinh nhưng không phải tất cả đều xảy ra trong toàn bộ vòng đời.

**5.Incremental Model – Mô hình tăng trưởng**

Example:



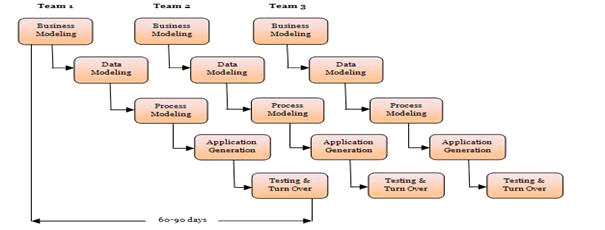
Diagram:



* Mô tả:
  + Trong mô hình này thì spec được chia thành nhiều phần.
  + Chu kỳ được chia thành các module nhỏ, dễ quản lý.
  + Mỗi module sẽ đi qua các yêu cầu về thiết kế, thực hiện, … như 1 vòng đời phát triển thông thường.
* Áp dụng:
  + Áp dụng cho những dự án có yêu cầu đã được mô tả, định nghĩa và hiểu một cách rõ ràng
  + Có nhu cầu về sản phẩm sớm
* Đặc điểm:
  + Ưu điềm:
    - Phần mềm làm việc một cách nhanh chóng trong suốt vòng đời phát triền
    - Mô hình này linh hoạt hơn, ít tốn kém hơn để thay đổi phạm vi và yêu cầu
    - Dễ dàng hơn trong việc kiểm tra và sửa lỗi với sự lặp lại nhỏ hơn
  + Nhược điểm:
    - Cần lập plan và thiết kế tốt
    - Cần một định nghĩa rõ ràng và đầy đủ của toàn bộ hệ thống trước khi nó có thể được chia nhỏ và được xây dựng từng bước
    - Tổng chi phí là cao hơn so với thác nước.

Tham khảo thêm: <https://melsatar.wordpress.com/2012/03/15/software-development-life-cycle-models-and-methodologies/>

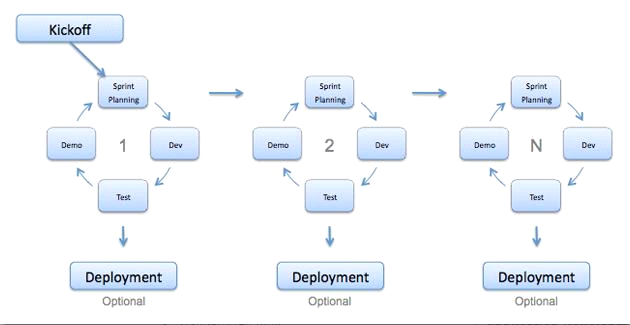
**6. RAD Model (Rapid Application Development)**



* Mô tả:
  + Là một dạng của incremental model
  + Trong mô hình RAD các thành phần hoặc chức năng được phát triển song song như thể chúng là các dự án nhỏ
  + Việc phát triển này theo thời gian nhất định, cung cấp và lắp ráp thành một nguyên mẫu làm việc
  + Điều này có thể nhanh chóng đưa ra một cái gì đó cho khách hàng để xem và sử dụng và cung cấp thông tin phản hồi liên quan đến việc cung cấp và yêu cầu của họ.
* Áp dụng:
  + RAD nên được sử dụng khi có nhu cầu để tạo ra một hệ thống có Modularized trong khoảng thời gian 2-3 tháng.
  + Nên được sử dụng khi đã có sẵn designer cho model và chi phí cao
  + Đặc điểm:
  + Ưu điềm:
    - Giảm thời gian phát triển.
    - Tăng khả năng tái sử dụng của các thành phần
    - Đưa ra đánh giá ban đầu nhanh chóng
    - Khuyến khích khách hàng đưa ra phản hồi
  + Nhược điểm:
    - Cần có một team giỏi để xác định yêu cầu phần mềm
    - Chỉ những hệ thống có module mới sứ dụng được mô hình này
    - Yêu cầu về dev/ design phải có nhiều kinh nghiệm
    - Phụ thuộc rất nhiều vào kỹ năng model

Tham khảo thêm: <http://istqbexamcertification.com/what-is-rad-model-advantages-disadvantages-and-when-to-use-it/>

**7. Agile Model**



* Mô tả:
  + Dựa trên mô hình iterative and incremental
  + Các yêu cầu và giải pháp phát triển dựa trên sự kết hợp của các function
* Áp dụng:
  + Nó có thể được sử dụng với bất kỳ loại hình dự án nào, nhưng nó cần sự tham gia và tính tương tác của khách hàng. Ngoài ra, nó có thể được sử dụng khi khách hàng yêu cầu chức năng sẵn sàng trong khoảng thời gian ngắn ( 3 tuần )
* Đặc điểm:
  + Ưu điểm:
    - Giảm thời gian cần thiết để tận dụng một số tính năng của hệ thống
    - Kết quả cuối cùng là phần mềm chất lượng cao trong thời gian ít nhất có thể và sự hài lòng của khách hàng
  + Nhược điểm:
    - Phụ thuộc vào kỹ năng của người phát triển phần mềmScalability
    - Tài liệu được thực hiện ở giai đoạn sau
    - Cần một team có kinh nghiệm Needs special skills for the team.

Tham khảo thêm: <http://istqbexamcertification.com/what-is-rad-model-advantages-disadvantages-and-when-to-use-it/>

1. **Viết trên nền tảng nào ?**

Dài vcl, xem qua link t đã gửi m =>>>>

[**http://notes.viphat.work/so-sanh-cac-ngon-ngu-lap-trinh-infographic?fbclid=IwAR38TynddoPej-Rp72MyH9Q36tAc5cGG6SM-\_\_jH5QWbyMOdIjAjm0zKHPc**](http://notes.viphat.work/so-sanh-cac-ngon-ngu-lap-trinh-infographic?fbclid=IwAR38TynddoPej-Rp72MyH9Q36tAc5cGG6SM-__jH5QWbyMOdIjAjm0zKHPc)



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Language*** | C++ | Csharp | Python |  |
| **Ưu điểm** | - Ngôn ngữ phổ biến, dễ tìm hiểu, dễ sử dụng.  - Thông dụng, phù hợp cho tất cả các hệ điều hành.  - Rất mạnh, nhiều thư viện kế thừa những điểm tốt của ngôn ngữ C.  - Dễ chuyển đổi với các ngôn ngữ khác. | - Được sự trợ giúp từ .Net framwork  - Dễ chuyển đổi với các ngôn ngữ: Java, Objective C, PHP, C++.  - Hướng đối tượng.  - Tốn ít chi phí, thời gian, kết quả đáng tin cậy  - Cải tiến các nhước điểm của C, C++,… | - Dễ học, dễ tìm hiểu, cú pháp easy + clearly.  - Có trên tất cả HĐH.  - Tương thích mạnh mẽ với Unix,hardware, thirty party  - Thư viện extensive, dễ build nhiều thứ.  - Có thể phát triên IOT. |  |
| **Nhược điểm** | - Rất khó + phức tạp, cần thời gian tìm hiểu, ở đây cũng không phải siêu khó, chỉ là khó hơn các ngôn ngữ thông dụng khác | - Cần trình độ vững vàng khi sử dụng ngôn ngữ này :v, không dành cho newbies  - Hiện tại chỉ chạy trên windows, chưa dùng cho MacOS hay những hđh khác, và yêu cầu cài đặt .Net Framework | - Có sự hạn chế với lập trình mobile  - Tốc độ chậm hơn các ngôn ngữ khác |  |

Thiệt thì t chưa thể xác định viết bằng ngôn ngữ nào luôn,