CẨM NANG

ĐỊNH HƯỚNG NGHỀ NGHIỆP IT

Homer Truong Le Hoang

Vietnam Top Software Professional Mentor

Talent Enterprise Connecting Hub on 3S System (TECH3S)

Homer Truong Le Hoang

Vietnam Top Software Professional Mentor

Canada Master of Information Technology (IT)

15-year experience in VN, AU, CA and US.

http://www.facebook.com/homertruong66

http://www.facebook.com/tech3s.mentor/

http://www.tech3s-mentor.com

http://www.linkedin.com/in/truonglehoang

Định hướng nghề nghiệp IT

Lời mở đầu

Chào mừng bạn đến với Thế giới Công nghệ thông tin (Information Technology – IT) !!!

Bạn đang đọc quyển cẩm nang này thì tôi cho rằng bạn đang ĐAM MÊ ngành IT hoặc chí ít cũng có hứng thú để tìm hiểu ngành này. Nếu giả thuyết của tôi đúng thì…CHÚC MỪNG BẠN đã có được điều kiện cần đầu tiên để bước vào thế giới IT, hoặc bạn chỉ tò mò muốn biết ngành IT thế nào thì tôi cũng hi vọng sau khi đọc xong cuốn cẩm nang này bạn sẽ ĐAM MÊ gia nhập ngành IT ☺

Ngành IT đã và đang phát triển như vũ bão, thay đổi thế giới loài người rất nhiều dù xuất hiện chưa lâu so với lịch sử loài người. Nói chung ngành IT khá là hay nhưng cũng vô cùng thách thức cho những ai theo nó. Cho nên, để đạt được thành công khi vào ngành IT thì họ cần phải có ĐAM MÊ thực sự, luôn chịu khó học hỏi cái mới, kiên trì, khổ luyện và luôn có định hướng rõ ràng các công việc họ sẽ theo đuổi ở những thời điểm khác nhau trong đời.

Tôi đến với ngành IT cũng thật là tình cờ. Thời tôi học cấp 3 trường Lê Hồng Phong TPHCM gần lúc thi đại học (1998) không được định hướng nghề nghiệp cũng như không có điều kiện tìm hiểu thị trường việc làm bên ngoài (do lúc đó thông tin việc làm trên internet còn hạn chế) nên tôi không hề biết mình thích ngành gì? nên học ngành gì? ra trường sẽ làm nghề gì? Vì thế lúc trường gửi danh sách xuống mỗi lớp cho học sinh chọn các trường & ngành sẽ thi đại học đợt 1,2,3 (năm 1998 được thi 3 trường) thì tôi chọn đại 3 trường & ngành dựa trên số đông trong lớp và thông tin "hot" do xã hội truyền miệng (thời Ba Má tôi thì có câu "Nhất Y, Nhì Dược, Được được Bách Khoa, còn thời tôi lúc đó thì "hot" là IT, Ngoại Thương rồi mới đến Y, Dược). Kết quả là tôi vô ngành IT của ĐH Bách Khoa TPHCM.

Sau khi bỏ ra 4.5 năm theo chương trình IT của ĐH Bách Khoa TPHCM, năm 2003 tôi ra trường mà không hình dung được mình sẽ làm công việc gì để mà tìm và khi đó các trang mạng tìm việc cũng chưa có mà tìm chủ yếu qua báo giấy, cho nên quá trình đi tìm việc của tôi hết sức khó khăn. Rồi khi tôi tham dự mấy buổi phỏng vấn đầu tiên mới phát hiện ra là mình đã học rất nhiều môn IT nhưng học rời rạc qua từng học kỳ (do không biết định hướng từ đầu) nên tôi không hiểu được sự liên hệ lẫn nhau giữa các môn đó. Điều này rất cần thiết để có thể tham gia vào dự án IT thực tế.

Vì tôi không hiểu các nhánh IT liên quan với nhau thế nào nên rớt phỏng vấn là dễ hiểu. Cũng đã vài lần tôi nhận được thư…cảm ơn của vài công ty sau khi tôi tham dự phỏng vấn !!!

Sau khi vật vã tìm việc 2-3 tháng mà chưa có kết quả, nói thật là tâm trạng tôi rất ư là thất vọng, suy nghĩ tiêu cực lóe lên: "mình chọn lầm ngành rồi sao? làm sao để có việc?", nhưng chợt nghĩ lại: "ủa, mình đâu có chủ động chọn ngành, mình đã để số phận đưa đẩy mà… ".
Vào một buổi sáng đẹp trời, buồn buồn tôi xách xe chạy tàn tàn dạo mát khu Quận 3 (thời ấy ít xe, ra đường mát lắm, không khí trong lành nữa). Vừa chạy vừa si nghĩ về nước Mĩ không biết sao tôi lại lạc vô đường Tú Xương, Quận 3. Đang loay hoay nhìn xung quanh định hướng thì bất chợt tôi nhìn thấy…bảng tuyển dụng "Nhân viên phần mềm" treo trước cửa công ty SaigonTel (bây giờ mấy bảng kiểu này đã trở thành huyền thoại).

Đắn đo một hồi tôi nghĩ thầm: "không lẽ mình ăn ở hiền nên ông bà thương dẫn đường mình tìm việc chăng?", dù gì đi nữa cứ vô hỏi thử, biết đâu hên. Thế là tôi mạnh dạn vô công ty hỏi thăm thì chú bảo vệ nói về chuẩn bị hồ sơ đi rồi lên nộp. Tôi liền chạy về nhà chuẩn bị một sấp hồ sơ giấy đem vô nộp.

Thật bất ngờ là hôm sau công ty hẹn tôi phỏng vấn buổi sáng thì buổi chiều nhận luôn…phù… tôi như trút một gánh nặng ngàn cân đeo tôi suốt 2-3 tháng trời… Thật ra lúc đó tôi không bị áp lực về tài chính phải có việc kiếm tiền liền (vì tôi đang là Kiện tướng Cờ Tướng Quốc Gia, Thành viên Đội tuyển Cờ Tướng TPHCM mỗi tháng đều có lương), nhưng tôi bị ức là đã bỏ công ra học đàng hoàng suốt 4.5 năm trời với kết quả…Khá xém Giỏi… mà lại không đủ khả năng làm việc thực tế là sao???
Khi vô dự án IT thực tế tôi mới tự tìm ra câu trả lời: **tôi thiếu quá nhiều kiến thức/kỹ năng mà trước đó tôi không có MENTOR định hướng giúp để chuẩn bị trước trong lúc còn ngồi trên giảng đường đại học**; tôi lấy được công việc đầu tiên cũng có phần may mắn là anh PGĐ Việt Kiều Mỹ, Chris Nguyen, đã phỏng vấn không quá gắt về chuyên môn như các công ty khác, và ghi nhận tính tình trung thực, chịu khó học hỏi của tôi.

Làm cho 2 công ty SaigonTel và TMA Solutions được 3 năm thì tôi đã qua thành phố Montreal, Canada để học Thạc sĩ về IT, nghiên cứu về Data Science trong 1 dự án thực tế của công ty Bell Canada và làm việc cho vài công ty Canada (tất cả đều là Tech Startups chứ không phải IT Outsourcing Companies) suốt 7 năm, đến năm 2012 thì tôi về lại TPHCM.

Sau khi về lại Việt Nam, tôi đã cùng vài bạn đã mở 1 số Tech Startups ở Việt Nam (SmartGuide tech product), Úc (SmartPush, SmartShip tech products), rồi làm Software Architect trong vài dự án của các Tech Startups ở Mỹ, rồi làm CTO/Co-Founder cho Tech Startup 4SV (http://4sv.vn), một nền tảng giúp Sinh Viên (SV) Việt Nam nâng cao kiến thức và kinh nghiệm khi đang học. Trong quá trình này, tôi đã phỏng vấn hàng trăm bạn SV IT mới ra trường thì…tôi bắt gặp…hình ảnh tôi 15 năm trước trong các bạn: **thiếu định hướng nghề nghiệp IT, thiếu kiến thức/kỹ năng để gia nhập dự án IT thực tế**. Tôi hơi bất ngờ vì trong tâm trí tôi nghĩ các bạn bây giờ có điều kiện nhiều hơn tôi hồi xưa thì phải tốt hơn tôi hồi trước chứ sao lại "u như kỹ" là y như cũ thế nhỉ???

Vì vậy với cuốn cẩm nang "Định hướng nghề nghiệp IT" này, tôi sẽ chia sẻ với các bạn đang/sẽ học IT những kiến thức/kinh nghiệm được đúc kết từ 15 năm trải nghiệm trong ngành IT của tôi ở Việt Nam, Canada, Úc, và Mỹ, với mong muốn giúp các bạn có định hướng nghề nghiệp IT phù hợp với mình, chuẩn bị kiến thức/kỹ năng tốt nhất để khi học xong tìm được việc làm phù hợp nhất cũng như tiến lên đỉnh cao nghề nghiệp: **Industry Rockstar** ☺

Cuốn cẩm nang gồm 3 phần chính:

- **Tổng quan ngành IT và các nhánh**: phần này giúp bạn có cái nhìn tổng quan về ngành IT, các nhánh của nó kết hợp với nhau ra sao để tạo thành 1 hệ thống IT hoàn chỉnh. Cuối mỗi mục trong một nhánh sẽ có phần tổng kết lại những điểm bạn cần nắm vững khi tìm hiểu nhánh này.

- **Các nghề IT phổ biến**: dựa trên các nhánh cấu thành nên hệ thống IT thì thị trường việc làm IT sẽ có các nghề IT tương ứng. Phần này cung cấp cho các bạn thông tin về các nghề IT phổ biến nhất trong thị trường việc làm. Đối với mỗi nghề sẽ cho bạn biết một số tiêu chí (Tính tính, Kiến thức, Kỹ năng) mà người theo nghề đó cần có để thành công nhất.

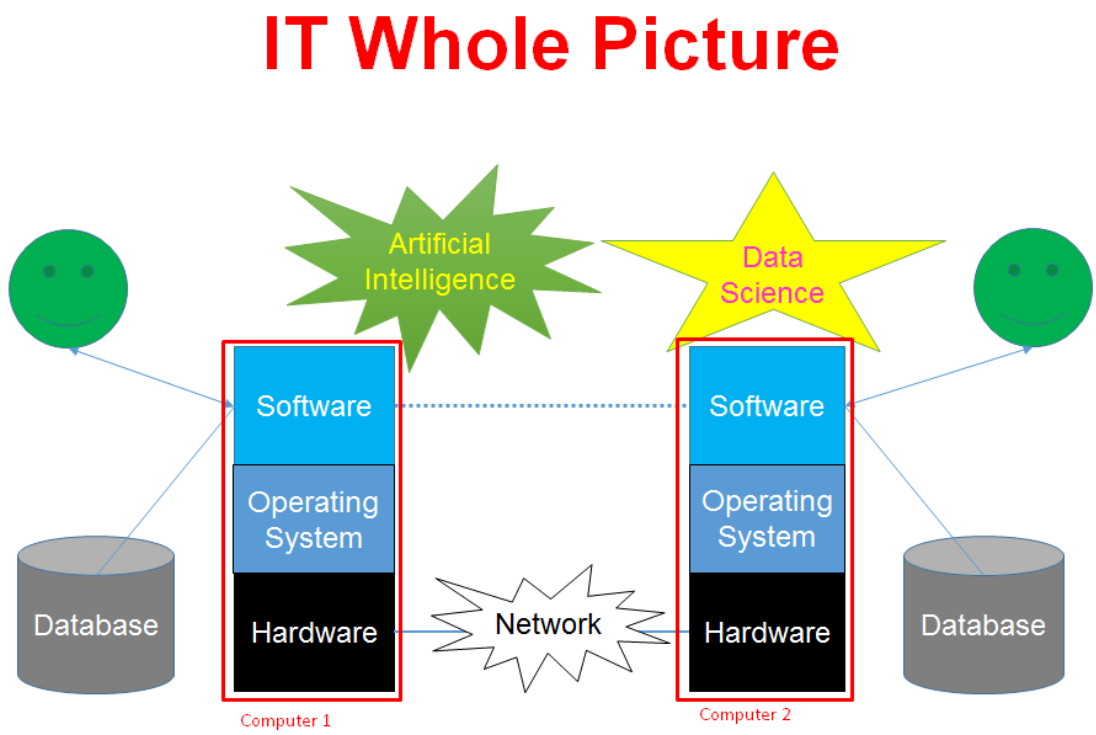
- **Căn bản lập trình**: trong các nghề IT phổ biến ở trên thì một số nghề đòi hỏi kỹ năng lập trình, các nghề còn lại không đòi hỏi kỹ năng lập trình nhưng nếu biết lập trình cũng giúp cho công việc tốt hơn, do đó phần này sẽ giúp bạn năm được một số điểm cản bản lập trình, tạo tiền đề cho bạn đi sâu hơn về lập trình khi cần.

Chúng ta bắt đầu khám phá Thế giới IT nhé…

I. Tổng quan ngành IT và các nhánh

Ngành IT rất là rộng lớn được chia thành nhiều nhánh khác nhau nhưng quan hệ mật thiết với nhau. Để thành công trong ngành IT thì việc nắm vững tổng quan ngành IT là điều cực kỳ quan trọng. Muốn vậy thì bạn phải hình dung ra bức tranh toàn cảnh IT.

Tôi đã tổng hợp được các nhánh của IT qua hình mô tả 1 hệ thống IT cơ bản dưới đây.

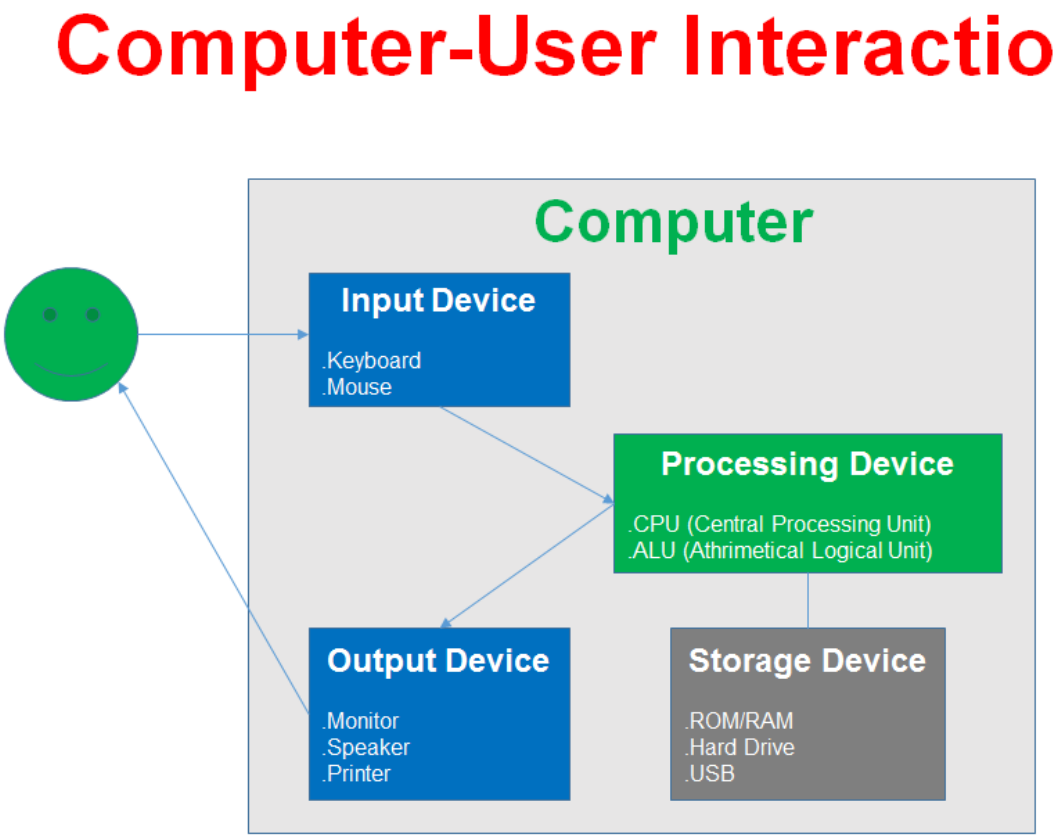


- Một **máy vi tính (Computer) – MVT** bao gồm 3 phần: **Phần cứng (Hardware)**, **Hệ điều hành (Operating System) – HĐH**, và **Phần mềm (Software)**.
- **Người dùng (User)** tương tác với MVT bằng phần mềm.
- Phần mềm thông qua HĐH điều khiển phần cứng thực thi các lệnh của người dùng.
- Phần mềm lưu **dữ liệu (Data),** khi người dùng tương tác với phần mềm qua thao tác tạo mới/chỉnh sửa/xóa, vào **Cơ sở dữ liệu (Database) – CSDL** để lần sau đọc lên lại cho người dùng tương tác tiếp.
- Trong quá trình phần mềm A xử lý yêu cầu của người dùng có thể cần dữ liệu được cung cấp bởi phần mềm B, A sẽ gọi B để lấy dữ liệu thông qua **mạng máy tính (Computer Network) – MMT.**
- Một số nhiệm vụ nguy hiểm cho con người khi làm thì **Trí tuệ nhân tạo (Artificial Intelligence – AI)** được sử dụng để chế tạo ra rô-bốt (robot) thay con người làm.
- Một số việc liên quan đến xử lý dữ liệu tốn rất nhiều thời gian cho người làm để đưa ra kết luận gì đó thì **Khoa học dữ liệu (Data Science)** sẽ giúp họ.

Ví dụ thực tế: 1 người X sử dụng phần mềm của ngân hàng VCB chuyển tiền cho 1 người Y ở ngân hàng ACB. **Giao dịch (transaction)** sẽ diễn ra như sau:

- Phần mềm VCB sẽ gọi phần mềm ACB thông qua MMT để lấy dữ liệu về tài khoản Y.
- Phần mềm VCB trừ tiền tài khoản của X trên VCB rồi lưu xuống CSDL của VCB.
- Phần mềm VCB gửi yêu cầu qua phần mềm ACB tăng tiền tài khoản của Y trên ACB.
- Phần mềm ACB tăng tiền tài khoản của Y trên ACB rồi lưu xuống CSDL của ACB.

Tiếp theo, chúng ta tìm hiểu tương tác giữa người dùng với MVT:



- Người dùng ra lệnh cho MVT qua **Thiết bị nhập (Input Device)** như **Bàn phím (Keyboard)**, **Chuột (Mouse)**…
- Lệnh của người dùng sẽ được **Thiết bị xử lý (Processing Device)** của MVT như **Bộ xử lý trung tâm (Central Processing Unit – CPU)** và **Đơn vị xử lý luận lý (Athrimetical Logical Unit – ALU)** xử lý rồi xuất kết quả qua **Thiết bị xuất (Output Device)** như **Màn hình (Monitor), Loa (Speaker), Máy in (Printer)**… để hiển thị cho người dùng xem.
- Những lệnh cần đọc/lưu dữ liệu sẽ được thiết bị xử lý đọc từ/lưu vào **Thiết bị lưu trữ (Storage Device)** như **Bộ nhớ chỉ đọc (Read Only Memory – ROM)**, **Bộ nhớ truy cập ngẫu nhiên (Random Access Memory – RAM), Đĩa cứng (Hard Drive)** …

Khi đã nắm được bức tranh tổng quan ngành IT ở trên, chúng ta bắt đầu tìm hiểu từng nhánh:

1. Phần cứng (Hardware)
2. Phần mềm (Software)
3. Cơ sở dữ liệu (Database)
4. Mạng máy tính (Computer Network)
5. Trí tuệ nhân tạo (Artificial Intelligence)
6. Khoa học dữ liệu (Data Science)

Cho dù bạn làm bất cứ nghề nào trong ngành IT thì ngoài kiến thức chuyên sâu của nhánh bạn đang làm thì cần kiến thức cơ bản của các nhánh khác nữa, nếu bạn biết sâu hơn mỗi nhánh khác đó càng nhiều càng tốt; điều này sẽ hỗ trợ công việc của bạn rất tốt bởi các nhánh này liên quan mật thiết với nhau trong 1 **hệ thống IT (IT System)**.

1) Phần cứng (Hardware)

Một MVT gồm 4 loại thiết bị chuẩn sau:

1. **Thiết bị nhập** : Bàn phím (Keyboard), Chuột (Mouse).
2. **Thiết bị xử lí** : CPU, ALU.
3. **Thiết bị xuất** : Màn hình (Monitor), Loa (Speaker), Máy in (Printer).
4. **Thiết bị lưu trữ** : Các loại đĩa (Disk), Bộ nhớ (RAM, ROM, Cache), USB.

MVT được chia thành một số loại:

- **MVT lớn (Main Frame)**: Dùng cho tổ chức lớn.
- **MVT vừa (Mini Frame)**: Tựa như MVT lớn nhưng sử dụng ở mức độ thấp.
- **MVT cá nhân (Personal Computer – PC)**: Dành cho user thông thường.
- **Siêu MVT (Super Computer)**: MVT được thiết kế tinh vi và phức tạp nhằm mục đích nghiên cứu khoa học và giải một số bài toán phức tạp có độ chính xác cao như máy đánh cờ vua Deep Blue.
- **MVT để bàn (Desktop)**: **Hộp máy (Case)** nằm ngang chứa màn hình hoặc đứng.
- **MVT xách tay (Laptop):** Có kích thước nhỏ gọn, màn hình được thiết kế bằng tinh thể lỏng úp lên hộp máy chính và trên hộp máy là sơ đồ bàn phím.
- **MVT cầm tay (Palmtop)**: Có kích thước rất nhỏ vừa trong lòng bàn tay.

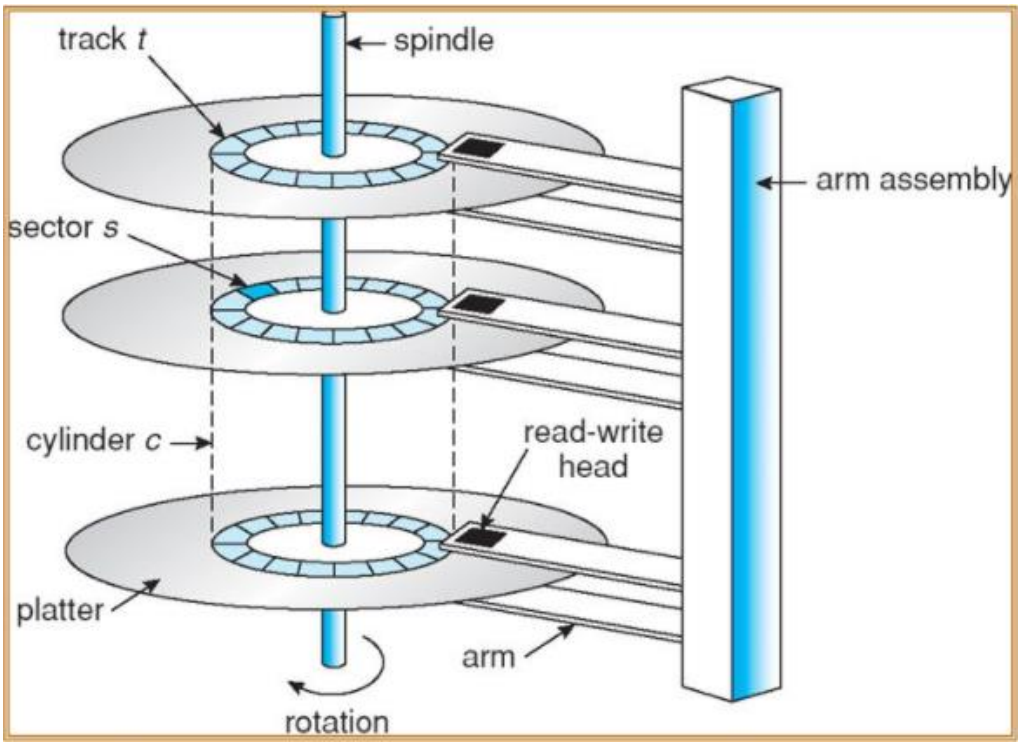
Một số thành phần chính cấu thành nên một MVT:

- **Bộ nguồn (Power):** Chuyển đổi điện xoay chiều của mạng điện thành điện một chiều.
- **Bo mạch chính (Mainboard):** Là mạch chính của toàn bộ hệ thống MVT.
- **CPU:** Điều khiển mọi hoạt động MVT.
  Tốc độ (Số xung nhịp trong 1 giây) = Tốc độ Bus x Ratio (Hz).
  - **Bộ đồng xử lí (Micro Co-Processor – MCP)** : Giúp CPU xử lí các phép toán dấu chấm động và các phép toán phức tạp khác.
  - **Đơn vị điều khiển (Control Unit – CU)** : Là trung tâm của CPU, điều khiển mọi công việc như lấy lệnh, giải mã lệnh, kích hoạt các bộ phận khác cùng xử lí.
  - **Mạch xung nhịp hệ thống (System Clock)** : Là mạch đánh nhịp để tạo ra các xung điện nhằm đồng bộ hóa các thao tác trao đổi thông tin, dữ liệu và tín hiệu của CPU với các thành phần bên ngoài.
  - **Đơn vị số học và luận lí (Arithmetical Logical Unit – ALU)** : thực hiện phép toán số học và luận lí.
- **Bộ nhớ chính (Main Memory):** Chứa chương trình và dữ liệu khi máy hoạt động.
  - **ROM (Read Only Memory)**: Bộ nhớ chỉ đọc, chứa các chương trình do hãng sản xuất cài sẵn.
  - **ROM BIOS (B**asic **I**nput **O**utput **S**ystem**)**: Nằm trên Mainboard chứa các chương trình nhập xuất cơ bản của hệ thống.

- **POST (Power On Self Test)**: Chương trình tự kiểm tra cấu hình hệ thống khi khởi động mà kết quả đem so sánh với các thông tin do người dùng khai báo trong **CMOS** và trả về thông báo lỗi (nếu có).
- **RAM (Random Acess Memory)**: Bộ nhớ có thể đọc/ghi dùng lưu dữ liệu khi máy hoạt động, toàn bộ dữ liệu trên RAM sẽ mất đi khi máy tắt (shutdown).
- **SRAM (Static RAM)**: Là RAM tĩnh tốc độ nhanh dùng làm bộ nhớ Cache.
- **DRAM (Dynamic RAM)**: Là RAM động, tốc độ chậm vì phải Refresh để bảo toàn điện tích, thường được dùng làm bộ nhớ chính cho MVT vì giá rẻ.
- **CMOS RAM**: Bộ nhớ chứa các thông tin về cấu hình do người dủng khai báo. Thông tin này không mất đi bởi được nuôi dưỡng bởi 1 nguồn pin. Để xoá thông tin trong CMOS RAM ta tháo nguồn pin này ra và ráp lại sau 5'.

- **Đĩa cứng (Hard Disk – HD)**
  - **Cấu trúc vật lí**: Một HD gồm nhiều xy-lanh (Cylinder - C), trên mỗi Cylinder gồm các mặt đĩa (Head - H) và trên mỗi Head được chia thành từng cung gọi là Sector (S - mỗi sector có dung lượng là 512B).

    Dung lượng đĩa = C x H x S x 512 (Bytes).



- **Cấu trúc luận lí**: HD có thể được chia thành nhiều **phân khu đĩa (partition)** gồm 2 loại là Primary Partition và Logical Partition.
  - **Primary Partition** là partition có thể chứa HĐH và có thể chứa chương trình để khởi động máy, mỗi HD có tối đa 4 primary partitions, trong đó phải có ít nhất 1 primary partition ở trạng thái "Active" để khởi động máy.
  - **Logical Partition** là các partition để chứa dữ liệu thuần, chúng được quản lí bởi **Extended Partition** (cũng là 1 primary partition).
- Sector đầu tiên của HD chứa **Master Boot Record** lưu thông tin về các partition. Các partition sẽ bắt đầu từ sector thứ 2 trở đi. Cấu trúc luận lí của mỗi partition :
  - **Directory** : Gồm nhiều điểm vào (entry), mỗi entry chiếm 32B chia làm 8 trường chứa thông tin về file (số entry hạn chế).
  - **Data** : có thể chứa các file chiếm các cluster cách quãng nhau không liên tục gọi là **sự phân mảnh đĩa (Disk Fragment)** làm cho sự truy xuất đĩa chậm. Khắc phục sự phân mảnh đĩa bằng phần mềm.

- **Thanh ghi (Register):** Là các thanh nhớ tạm thời dùng để lưu trữ và xử lí dữ liệu.
- **Bộ nhớ đệm của CPU (CPU Cache):** Là bộ nhớ tạm thời nằm trong CPU (Cache nội L1) hay nằm trên Mainboard (Cache ngoại L2) làm tăng tốc độ truy xuất của máy.
- **Kênh dẫn (Bus)**
  - **Bus điều khiển (Control Bus)**: Là các kênh dẫn dùng để truyền các tín hiệu điều khiển của CPU đến các thiết bị khác.
  - **Bus dữ liệu (Data Bus)**: Là các kênh dẫn song song nhằm thực hiện việc liên lạc nội bộ giữa các bộ phận bên trong CPU. Độ rộng và tốc độ Bus góp phần quyết định tốc độ hoạt động của CPU.
  - **Bus địa chỉ (Address Bus)**: Là các kênh dẫn gồm nhiều dây dẫn song song để xác định địa chỉ các ô nhớ. Độ rộng của Bus địa chỉ sẽ quyết định dung lượng bộ nhớ RAM cực đại mà CPU quản lí được.
- **Các vi mạch hỗ trợ (Chipset)**
  - **BIOS (Basic Input Output System)**: Hệ thống nhập xuất cơ bản.
  - **KbC (Keyboard Controller)**: Bộ điều khiển bàn phím.
- **Cổng kết nối (Port):** Là các ổ cắm để kết nối các thiết bị ngoại vi với Mainboard.
  - **Cổng bàn phím, chuột**: COM, USB
  - **Cổng màn hình**: VGA, HDMI
  - **Cổng máy in:** LPT

Hệ cơ số được sử dụng trong MVT là **hệ nhị phân (Binary System)** gồm hai con số là 0 và 1. Mỗi con số 0 hoặc 1 gọi là 1 **bit** (**b**). **Byte** (**B**) là đơn vị chính được dùng trong MVT. Một **Byte gồm 8 bit** có giá trị thập phân từ 0-255, ví dụ: Byte 11111100 có giá trị là 252.

Các đơn vị tính lớn hơn Byte :

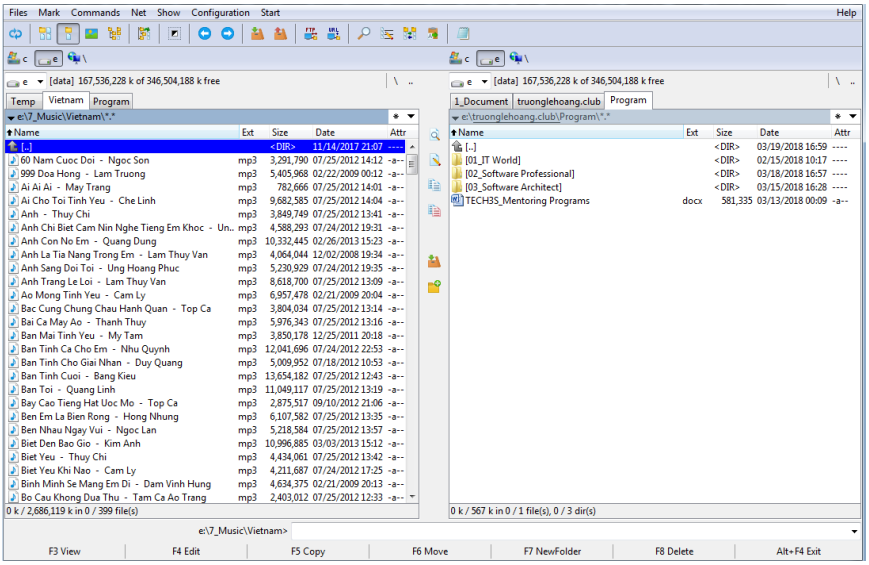
- **1KB (Kilo Byte)** = 2^{10} Bs = 1024 Bs
- **1MB (Mega Byte)** = 2^{10} KBs = 1024 KBs
- **1GB (Giga Byte)** = 2^{10} MBs = 1024 MBs
- **1TB (Tetra Byte)** = 2^{10} GBs = 1024 GBs

Đơn vị cơ bản lưu dữ liệu trong MVT là **tập tin (File)**, đơn vị lớn hơn File là **thư mục (Directory)**. Một thư mục có thể chứa nhiều File. Một File có nhiều tính chất, hai tính chất quan trọng của File là **tên (name)** và **thuộc tính (attribute)**. Dung lượng File được tính bằng đơn vị Byte. Tên file gồm có 2 phần:

1. **Phần tên chính**: Dùng để xác định File.
2. **Phần mở rộng (Đuôi File)**: Có chiều dài tối đa là 3 kí tự, dùng để xác định loại File.

Ví dụ: **readme.txt**: File dữ liệu; **report.doc**: File văn bản; **config.sys**: File hệ thống.

Trên Windows, dùng phần mềm **Total Commander** để quản lý files tiện lợi:



Các điểm bạn cần nắm vững trong mảng Phần cứng

1. Các thiết bị của MVT
2. Cách người dùng tương tác với MVT thông qua các thiết bị
3. Các thành phần chính cấu thành MVT
4. Đơn vị đo lường trong MVT
5. Quản lý thư mục/tập tin trong MVT

2) Phần mềm (Software)

Trên đây chúng ta vừa tìm hiểu nhánh phần cứng của ngành IT. Chỉ với phần cứng không thì người dùng không thể tương tác với MVT được. Do đó cần một cách thức để người dùng tương tác được với MVT: đó là **phần mềm ứng dụng (Application Software)**, gọi tắt là phần mềm (khác với phần mềm hệ thống (System Software) không có tương tác người dùng).

Phần mềm là các chương trình chạy trên MVT giúp người dùng làm những chức năng nào đó phục vụ cho công việc của họ. Ví dụ:

- Phần mềm **Microsoft Word** giúp ta soạn văn bản.
- Phần mềm **Media Player** giúp ta nghe nhạc.
- Phần mềm **Adobe Photoshop** giúp ta thiết kế hình ảnh.
- Phần mềm **Google Chrome** giúp ta lướt các trang web.

Phần mềm nguồn mở (Open-Source Software) – PMNM là những phần mềm được xây dựng, sử dụng, phân phối, sửa đổi theo những **nguyên tắc nguồn mở (Open-Source Rules)**:

- Quyền sao chép phần mềm và phân phối các bản sao chép.
- Quyền truy nhập nguồn để có thể cải tiến phần mềm theo ý mình, những phần cải tiến này cũng không được giấu mà phải mở để mọi người biết.

Phần mềm nguồn đóng (Closed-Source Software) – PMNĐ là những phần mềm mà **mã nguồn (Source Code)** của công ty/cá nhân viết ra không được công khai cho công chúng biết.

Các PMNM thường chạy trên nền **Linux, cũng là một HĐH nguồn mở** trong khi các PMNĐ thường chạy trên nền Microsoft Windows.

Linux được phát triển từ **UNIX (HĐH chủ yếu dành cho các máy server)**, nó thừa hưởng các điểm mạnh của UNIX như tính bảo mật cao, hướng mạng, ổn định…và đặc điểm đặc biệt của riêng nó là có thể chạy trên máy đơn PC không cần qua môi trường mạng.

Các nhà phát triển Linux đang cố gắng xây dựng hệ thống Linux ngày càng gần gũi với người dùng giống như hệ thống MS Windows. Không giống như MS Window chỉ do một mình Microsoft sản xuất, Linux do nhiều nhà sản xuất tạo ra nên rất đa dạng.
Tuy nhiên, cái chung nhất là tạo ra các PMNM chạy trên Linux càng thân thiện và càng tương thích với MS Windows càng nhiều càng tốt.
Ví dụ: Bộ **OpenOffice** tương thích hoàn toàn với bộ **MS Office**. Các dữ liệu đã soạn bằng Microsoft Office đều có thể chạy lại trên OpenOffice.

So sánh ưu/nhược điểm giữa PMNM & PMNĐ:

Ưu điểm	Nhược điểm
1. Giá rẻ 2. Độc lập: không bị lệ thuộc nhà cung cấp nào 3. Bảo đảm an toàn và riêng tư: không có **"hộp đen" (black box)** và **"cửa hậu" (back door)** 4. Tích thích ứng cao: sửa đổi dễ dàng để thích ứng với yêu cầu mới 5. Tuân thủ các chuẩn 6. Không bị hạn chế về sử dụng 7. Tính lâu dài: được phát triển bởi "everyone" 8. Phát triển dễ dàng	1. Chưa có hỗ trợ kĩ thuật tin cậy: Về mặt pháp lí thì không ai có nghĩa vụ bắt buộc phải cung cấp các dịch vụ hỗ trợ PMNM. 2. Không có bảo đảm rằng một phát triển cụ thể nào đó sẽ thực hiện được. 3. Năng lực hạn chế của người sử dụng, nhất là trong các cơ quan công quyền.

HỆ ĐIỀU HÀNH

HĐH là một phần mềm hệ thống điều khiển mọi hoạt động của MVT, là thành phần trung gian giúp phần mềm giao tiếp với phần cứng. Nó gồm 3 thành phần chính:

1. **Phần lõi (Kernel)** : quản lý các thành phần của MVT.
2. **Phần dịch vụ (Service)** : cung cấp các dịch vụ cho User.
3. **Phần giao diện (Interface)** : cung cấp giao diện giao tiếp với User.

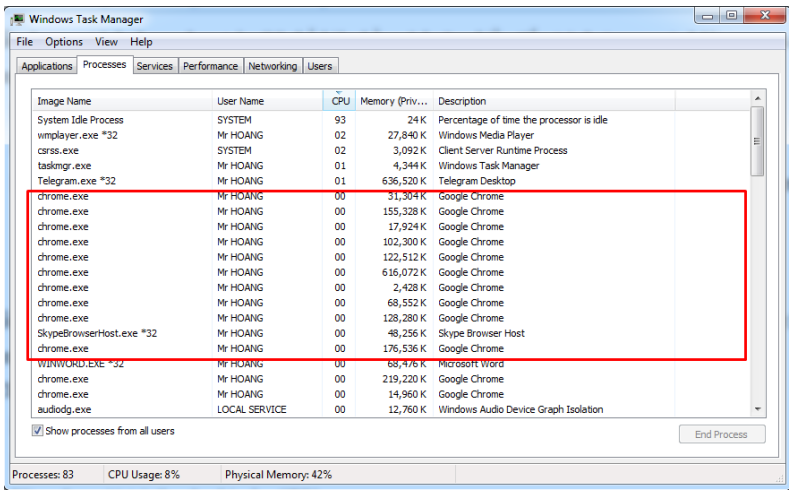
Lịch sử phát triển của HĐH:

- **Batch Processing**: Xử lí theo **lô (Batch)**, mỗi lần nạp lên RAM một chương trình, chạy xong mới nạp tiếp chương trình khác.
- **Multi Programming**: Xử lí đa chương trình, mỗi lần nạp nhiều chương trình lên RAM, xử lí các chương trình theo một thứ tự nào đó nhưng mỗi lần chỉ chạy một chương trình và chạy cho đến hết mới chạy tiếp chương trình khác.
- **Multi Tasking**: Xử lí đa nhiệm, mỗi lần nạp nhiều chương trình lên RAM, xử lí đồng thời các chương trình theo một thứ tự nào đó và định thời cho từng chương trình.
- **Multi Processing**: Xử lí song song, nhiều CPU cùng chạy trên cùng một MVT.
- **Multi Computer**: Xử lí song song với nhiều MVT xử lí cùng lúc.

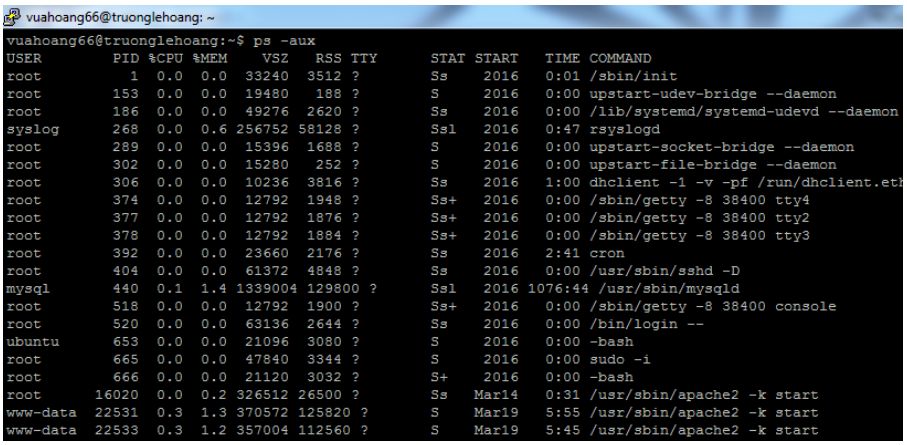
Mỗi chương trình đang chạy trên OS thể hiện bằng 1/nhiều **tiến trình (process)**. Mỗi process có 3 trạng thái:

- **Ready** : Đang chờ để được chạy
- **Running** : Đang chạy
- **Blocked** : Đang bị chặn để chờ **nhập xuất (I**nput/**O**utput **– IO)** hay tương tác phía User.

Trên Windows, mở Task Manager để xem processes:



Trên Linux, chạy lệnh ps –aux để xem processes:



Mỗi Process được điều khiển bởi 1 **bộ điều khiển tiến trình (Process Control Block – PCB)**. Một PCB là một cấu trúc dữ liệu do OS cấp để quản lý Process.

PCB chứa các thông tin về process gồm có:

- **Danh định của process**
- **Bộ đếm chương trình**
- **Vùng lưu giá trị thanh ghi của CPU**
- **Độ ưu tiên của process**

Khi trong hệ thống có nhiều process thì cần **định thời (schedule)** cho các process tức là chọn thứ tự thực hiện các process sao cho việc sử dụng CPU hiệu quả nhất.

Một số mục tiêu định thời:

- **Công bằng giữa các process.**
- **Cực đại số process phục vụ trong 1 đơn vị thời gian.**
- **Tránh lãng phí tài nguyên…**

Có 3 cấp định thời:

- **High – Level** : Xác định thời điểm tạo process.
- **Middle – Level** : Xác định thời điểm process có thể tham gia cạnh tranh CPU với các process khác.
- **Low – Level** : Chọn process trong **hàng đợi (Ready Queue)** để gán cho CPU xử lí.
- Việc định thời được thực hiện bởi **bộ định thời và chuyển phát (Scheduler and Dispatcher)**.

Một số giải thuật định thời:

- **FIFO (F**irst **I**n **F**irst **O**ut**)**: process nào đến trước được xử lí trứơc.
- **SJF (S**hortest **J**ob **F**irst**)**: process nào có thời gian cần xử lí nhỏ nhất được xử lí trước.
- **SRT (S**hortest **R**emaining **T**ime**)**: process có thời gian cần xử lí còn lại nhỏ nhất được xử lí trước.
- **Round Robin** : Chia thời gian thành những đoạn bằng nhau gọi là quantum time q. Mỗi process được xử lí trong một khoảng thời gian q, nếu hết thời gian q mà process chưa kết thúc thì process sẽ được đưa xuống cuối Queue để chờ đợt xử lí sau.

Trong quá trình thực thi, các process có thể tranh chấp tài nguyên hệ thống có thể dẫn đến trì hoãn. Do đó, một vấn đề mà OS cần phải làm là giải quyết tranh chấp.
Có 3 phương pháp giải quyết tranh chấp:

- **Dùng giải thuật phần mềm.**
- **Dùng phần cứng có hỗ trợ các lệnh đặc biệt.**
- **Dùng biến Semaphore được quản lý bởi OS.**

HĐH có các chiến lược quản lý **bộ nhớ (RAM)**:

- **Chiến lược nạp (Fetch)**: Xác định thời điểm lấy data or program nạp vào RAM từ Disks, nạp theo yêu cầu hoặc có dự đoán trước.
- **Chiến lược đặt (Place)**: Gồm 3 giải thuật là First Fit (Chọn vùng nhớ trống đầu tiên), Best Fit (Chọn vùng nhớ trống vừa khít) và Worst Fit (Chọn vùng nhớ trống lớn nhất).
- **Chiến lược cấp phát (Allocate)**: Cấp phát liên tục hoặc rời rạc.

Mỗi truy cập trên **đĩa cứng (Hard Disk)** của OS gồm 3 thao tác:

- **Seek** : Tìm sector cần truy cập.
- **Rotate** : Quay đĩa đến vị trí sector đó.
- **Transfer** : Truyền data.

HĐH định thời truy cập đĩa cứng bằng các giải thuật sau:

- **FCFS (F**irst **C**ome **F**irst **S**erved**)**: Truy cập đến trước được phục vụ trước.
- **SSTF (S**hortest **S**eek **T**ime **F**irst**)**: Truy cập có thời gian truy cập nhỏ nhất phục vụ trước.
- **Scan**: Đầu đọc/ghi dịch chuyển theo 1 chiều nhất định và chỉ đổi hướng khi không còn truy cập nào phía trước.
- **N-Step Scan**: Giống Scan nhưng các truy cập đến sau sẽ phục vụ theo chiều ngược lại.

Vì dung lượng bộ nhớ chính có hạn mà số lượng process được thực hiện rất nhiều như ta vừa nghe nhạc vừa gõ văn bản vừa chạy một mớ các chương trình khác… nên cần phải tạo ra **bộ nhớ ảo (Virtual Memory)** để giải quyết vấn đề trên.

Đây là kĩ thuật tách rời địa chỉ mà process truy cập và địa chỉ của bộ nhớ chính. Địa chỉ mà process truy cập được ánh xạ từ đĩa cứng đóng vai trò là bộ nhớ ảo lên RAM khi cần.

Phép ánh xạ sẽ thực hiện trên từng khối (block). Có 3 phương pháp chia:

- **Phân trang (Paging)**: Chia bộ nhớ ảo thành các khối có kích thước bằng nhau. Mỗi khối có một bit gọi là **Resident Bit** để xác định trang đã được ánh xạ (=1) hay chưa (=0: Default), một địa chỉ của khối trên đĩa cứng và một địa chỉ của khối trên RAM sẽ được ánh xạ lên.
- **Phân đoạn (Segmentation)** : Chia bộ nhớ ảo thành các khối có kích thước khác nhau
- **Kết hợp 2 pp trên**: Phân đoạn trước rồi phân trang trong mỗi đoạn.

Cơ chế truy cập **IO (Input/Output)** của process:

- **Blocking** : process chỉ xử lí tiếp khi thao tác IO hoàn tất
- **Non-Blocking** : process có thể xử lí tiếp ngay lập tức dù thao tác IO xong hay chưa.

Các chiến lược quản lý IO

- **Buffering**: dành RAM ltrữ data tạm thời cho việc truyền data.
- **Caching**: Write Back (Flush) và Write Through.
- **Spooling (cho máy in)**: Lưu trữ output của process và định thời xuất ra thiết bị sau đó.

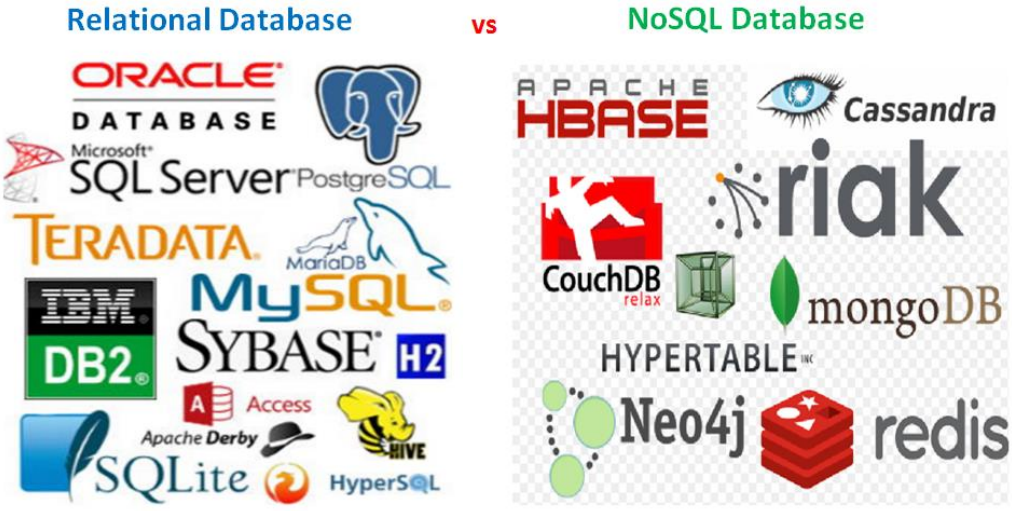
Các điểm bạn cần nắm vững trong mảng Phần mềm

1. Phần mềm là gì.
2. PMNM vs PMNĐ.
3. HĐH là gì.
4. Cách HĐH quản lý tiến trình.
5. Cách HĐH quản lý bộ nhớ.
6. Cách HĐH quản lý đĩa cứng.
7. Cách HĐH quản lý IO.

3) Cơ sở dữ liệu (Database)

Đến đây thì ta đã biết được người dùng tương tác với MVT qua phần mềm. Trong quá trình tương tác thì dữ liệu do người dùng tạo ra như sản phẩm (Product), đơn hàng (Order) được lưu tạm thời trên RAM, khi tắt máy sẽ mất. Vì vậy cần phải có chỗ để lưu dữ liệu này lâu dài.

CSDL là một hệ thống lưu dữ liệu có tổ chức để truy xuất dữ liệu tiện lợi dùng **ngôn ngữ truy vấn có cấu trúc (Structured Query Language – SQL)** hơn là lưu dữ liệu dưới dạng tập tin thông thường. Chính vì thế CSDL được dùng để lưu dữ liệu cho các phần mềm.
Hệ quản trị cơ sở dữ liệu (DataBase Management System – DBMS) là hệ thống giúp chúng ta quản trị CSDL thuận tiện và dễ dàng.



CSDL có 2 loại:

- **CSDL quan hệ (Relational Database)**: Là hệ thống lưu dữ liệu trong các **bảng (tables)**, các bảng có các **quan hệ (relationships)** với nhau. CSDL quan hệ được thể hiện với **mô hình quan hệ thực thể (Entity Relationship Diagram – ERD).** Dữ liệu được tạo/cập nhật/truy xuất/xóa dùng SQL.
- **CSDL NoSQL (NoSQL Database)**: Là hệ thống lưu dữ liệu không dùng bảng và quan hệ như CSDL quan hệ mà dùng mô hình Key – Value để lưu và truy xuất dữ liệu rất nhanh.

Tùy theo lĩnh vực và yêu cầu của phần mềm chúng ta đang phát triển mà ta sẽ chọn một trong hai loại CSDL trên.

Các điểm bạn cần nắm vững trong mảng CSDL

1. CSDL là gì.
2. Phân biệt 2 loại CSDL: CSDL Quan hệ & CSDL NoSQL và cách sử dụng 2 loại này.

4) Mạng máy tính (Computer Network)

Với phần cứng, HĐH, phần mềm và CSDL thì người dùng đã có 1 hệ thống IT đủ để thực hiện các chức năng họ mong muốn trên 1 MVT.

Tuy nhiên có những phần mềm chạy trên 1 MVT cần truy xuất dữ liệu trên 1 MVT khác như ví dụ VCB & ACB ở trên thì cần phải có MMT để thực hiện việc này.

Các MVT muốn giao tiếp với nhau thì phải nối mạng với nhau và dựa trên 1 **giao thức (Protocol)** đã được định nghĩa nào đó. Một MVT đóng vai trò **máy chủ (Server)** cung cấp các dịch vụ để các MVT khác đóng vai trò **máy khách (Client)** kết nối tới sử dụng dịch vụ.

Dựa trên 2 vai trò này mà giới chuyên gia đã đưa ra mô hình tương tác huyền thoại trên MMT: **Mô hình Client-Server**.

Mạng máy tính có các qui mô sau:

- **Mạng cá nhân (Personal Area Network – PAN)**: hay Bluetooth xài khoảng cách gần.
- **Mạng nội bộ (Local Area Network – LAN)**: xài cho công ty/gia đình trong phạm vi nhỏ.
- **Mạng nội thành (Metropolitant Area Network – MAN):** xài cho 1 thành phố.
- **Mạng diện rộng (Wide Area Network – WAN)**: xài cho 1 tỉnh/quốc gia.
- **Mạng toàn cầu (Internet)**: kết nối cả thế giới.

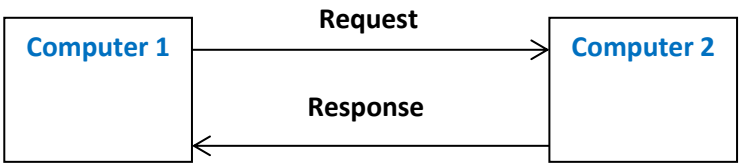
Hai thành phần quan trọng trong mạng:

- **Phần cứng (Hardware)**:
  - MVT - Hosts (Personal Computer)
  - Card mạng (NIC - Network Interface Card)
  - Phương tiện truyền (Medium)
  - Bộ khuếch đại (Hub)
  - Bộ chuyển mạch (Switch)
  - Bộ tìm đường (Router)
- **Phần mềm (Software)**:
  - HĐH mạng (Network Operating Sytem)
  - Phần mềm mạng (Network Software)

Lịch sử của mạng máy tính:

- Năm 1969, mạng đầu tiên ra đời là mạng APARNET.
- Năm 1974, một mạng khác ra đời là NFSNET và bắt đầu từ đây chuẩn **TCP/IP** ra đời.
- Ngày 1/1/1983, APARNET & NFSNET sáp nhập lại thành một mạng duy nhất vẫn lấy tên là APARNET.
- Đến năm 1991 thì mạng **Internet** chính thức ra đời còn APARNET chuyển thành mạng MILLINET phục vụ cho quân đội Mỹ.

Giao thức là một tập hợp các câu hỏi/đáp giữa 2 MVT giúp chúng nói chuyện với nhau. Giao thức định dạng các câu hỏi/đáp và trình tự thực hiện các câu hỏi đáp đó.
Giao thức được sử dụng trên Internet là **giao thức hỏi/đáp (Request/Response Protocol)**.



Xây dựng một ứng dụng mạng tức là viết 2 chương trình:

- Một chương trình **Server.**
- Một chương trình **Client**.

Để cho 2 chương trình này giao tiếp với nhau ta phải xây dựng một giao thức cho chúng.

Một số giao thức của các ứng dụng mạng cơ bản đã được định nghĩa:

- Truyền nhận file **(File Transfer Protocol – FTP)**: chạy trên port 21
- Gửi Email **(Simple Mail Transfer Protocol – SMTP)**: port 25
- Nhận Email offline **(Post Office Protocol – POP)**: port 110
- Nhận Email online **(Internet Message Access Protocol – IMAP)**: port 143
- Web **(Hyper Text Transfer Protocol – HTTP)** : port 80

Ứng dụng trên nền Web (Web Application) là một ứng dụng mạng dựa trên mô hình Client-Server nhưng nới rộng thành Mô hình ứng dụng Web 3 lớp **(3-tier Web Application Model)**, trong đó:

- **Trình duyệt** (**Web Client or Web Browser**) yêu cầu các trang Web từ Web Server.
- **Web Server** sẽ xử lí các yêu cầu này rồi trả kết quả về cho Web Browser. Trong quá trình xử lí, Web Server có thể truy xuất **Database Server** để lấy/cập nhật thông tin.

Xây dựng một ứng dụng web là xây dựng các trang web tương tác với người dùng và xây dựng các **nghiệp vụ phía máy chủ** (**Server-Side logic)** xử lí các tương tác này. Ví dụ: xây dựng 1 trang **web thương mại điện tử (e-commerce web)** để người dùng chọn **hàng (product)**, lập **đơn hàng (order)** thì nghiệp vụ cần xây dựng phía máy chủ là quản lý sản phẩm, xử lý đơn hàng, quản lý kho hàng...

Trang Web được xây dựng bằng **ngôn ngữ liên kết văn bản (Hyper Text Markup Language – HTML)** có thể được nhúng thêm các đoạn mã viết bằng ngôn ngữ **JavaScript** để làm sinh động thêm trang Web hoặc tương tác với người dùng.

Định dạng 1 trang WEB:

```
<html>
  <head>
    <title> Web Title </title>
    <script>
      alert('Hello Web with Javascript');          // đoạn mã Javascript
    </script>
  </head>
  <body>
    Web Content
  </body>
</html>
```

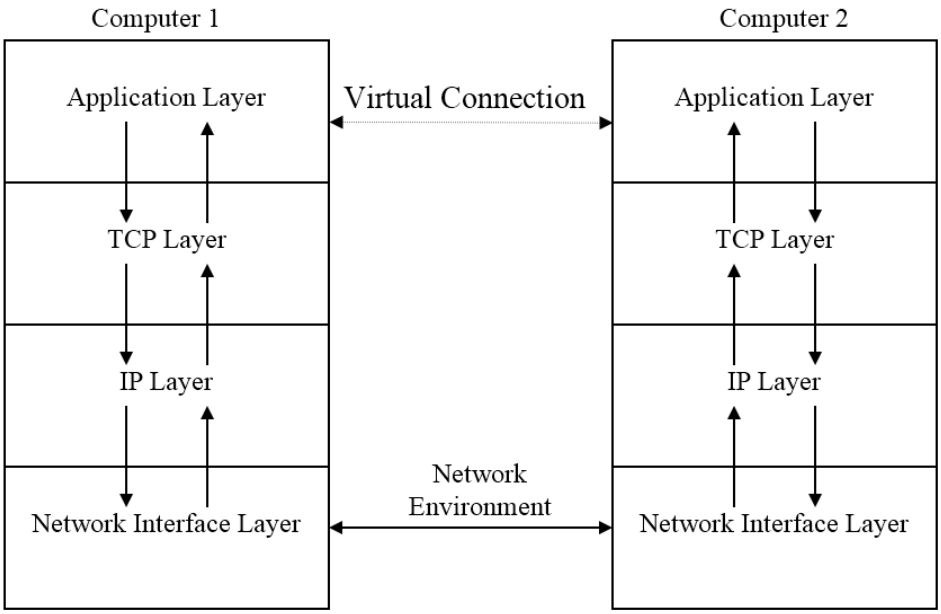
Mọi thứ nằm trong cặp tag <body> … </body> là nội dung trang Web sẽ được thể hiện.

Server-side logic xử lý các tương tác với người dùng có thể được viết bằng các ngôn ngữ lập trình phổ biến như Java, .Net, PHP…

Mô hình lý thuyết của mạng máy tính dựa trên mô hình **OSI (Open System Interconnect)** 7 lớp (Layer):

- **Physical Layer**: Tầng này làm nhiệm vụ thực hiện kết nối đường truyền vật lí giữa 2 MVT. Do đó chúng ta cần quan tâm đến 3 vấn đề: Phương tiện truyền, Chuẩn giao tiếp và Thông số đường truyền. Phương tiện truyền: gồm 2 loại
  - **Có dây (Wire):** cáp quang (Fiber Optic) or cáp đồng (Cooper Cable)
  - **Không dây (Wireless)**: ra di ô (Radio) or vệ tinh (Satellite)
- **Datalink Layer**: Tầng này làm nhiệm vụ truyền data giữa 2 máy cho trước và có điều khiển lỗi. Như vậy nó cần phải có 2 module: module truyền (**Transferring Module**) và module điều khiển lỗi (**Error Controlling Module**).
- **Network Layer**: Tầng này có nhiệm vụ truyền data giữa 2 máy bất kì trong mạng và có điều khiển lỗi. Nó cần phải có 2 module: tìm đường **(Routing)** và chuyển mạch **(Switching)**.
- **Transport Layer** : Tầng này có nhiệm vụ cung cấp giao tiếp giữa 2 ứng dụng trên 2 MVT và đánh địa chỉ tường minh (địa chỉ port). Quản lý kết nối bằng giải thuật bắt tay 3 lần.
- **Session Layer** : Tầng này có nhiệm vụ thiết lập, quản lí và kết thúc các phiên làm việc giữa các ứng dụng và dò tìm vị trí các dịch vụ.
- **Presentation Layer**: Tầng này có nhiệm vụ mã hóa data, nén data và đảo ngược data.
- **Application Layer**: Cung cấp các dịch vụ Internet
  - Bảo mật trên mạng: Dùng các phương pháp mã hoá.
  - Xác định tên miền dựa vào DNS (Domain Name Server)
  - Quản trị: Dùng SNMP (Simple Network Management Protocol)
  - Cung cấp các dịch vụ phổ biến: FTP, Email, WWW…

Mô hình mạng máy tính thực tế dựa trên 1 mô hình thu gọn của OSI, gọi là **mô hình TCP/IP** có 4 lớp.



Tương tác giữa Computer 1 & Computer 2 ở hình trên sẽ diễn ra như sau:

- User trên Computer 1 ở lớp ứng dụng gởi yêu cầu 1 trang WEB trên Computer 2 thì yêu cầu đó được chuyển xuống các lớp dưới để thêm phần header & tailer và sau đó được truyền đi bằng các tín hiệu vật lí bởi lớp Network Interface sang lớp Network Interface của Computer 2 để gởi lên lớp Application của Computer 2.
- Tương tự, trang WEB từ Computer 2 sẽ được gởi đến cho Computer 1 theo yêu cầu.
- User chỉ thấy Virtual Connection giữa 2 máy, thực tế thì yêu cầu phải đi theo trình tự như trên.

MVT chúng ta muốn truy cập thông tin trên internet thì phải đăng ký với **nhà cung cấp dịch vụ internet** (**Internet Service Provider – ISP)**. Mỗi ISP có nhiều khách hàng và có nhiều loại dịch vụ Internet khác nhau.

Để việc trao đổi thông tin trong mạng Internet thực hiện được, mỗi máy trong mạng cần phải được định danh để phân biệt với các máy khác. Mỗi MVT trong mạng được định danh bằng một nhóm các số được gọi là **địa chỉ IP (IP Address)** gồm 4 số nguyên có giá trị từ 0 đến 255 và được phân cách nhau bằng dấu chấm.

Ví dụ : 205.25.15.23, 192.168.10.20

Địa chỉ IP có giá trị duy nhất trong toàn mạng Internet. Ủy ban phân phối địa chỉ IP của thế giới sẽ phân chia các nhóm địa chỉ IP cho các quốc gia khác nhau. Thông thường địa chỉ IP của một quốc gia do các cơ quan bưu điện quản lý và phân phối lại cho các ISP. Một MVT khi thâm nhập vào mạng Internet cần phải có một địa chỉ IP. Địa chỉ IP này có thể được cấp tạm thời hay cấp vĩnh viễn.

Thông thường, các máy Client kết nối vào Internet thông qua một ISP bằng đường điện thoại hoặc đường dây cáp. Khi kết nối, ISP sẽ cấp tạm một địa chỉ IP cho máy Client. Như vậy, các máy Client không cần phải xin cung cấp một địa chỉ IP riêng lẽ.

Để việc phân phối địa chỉ của các cơ quan quản lý địa chỉ IP được dễ dàng, các địa chỉ IP từ 0.0.0.0 đến 255.255.255.255 (32 bit) được phân thành các lớp (class) A, B, C và D. Một lớp gồm một số lượng các địa chỉ IP. Số lượng địa chỉ IP ở các lớp là khác nhau.

Một địa chỉ IP gồm 2 phần: NetID & HostID. Các máy trên cùng một mạng sẽ có NetID giống nhau.
Class A: 1.0.0.0 - 127.255.255.255 (0xxxxxxx : 8bit NetID)
Class B: 128.0.0.0 - 191.255.255.255 (10xxxxx.. : 16bit NetID)
Class C: 192.0.0.0 - 223.255.255.255 (110xxxx.. : 24bit NetID)
Class D: lớp địa chỉ dùng cho multicast.
Class E: lớp địa chỉ để dành.
Các địa chỉ IP không có trên Internet dùng để cấp cho các máy trong các mạng cục bộ (Local Area Network):
Class A : 10.0.0.0 - 10.255.255.255
Class B : 172.16.0.0 - 172.31.255.255
Class C : 192.168.0.0 – 192.168.255.255

Khi truyền thông tin, một máy cần phải biết địa chỉ IP của máy nhận có trong cùng một mạng với mình không. Để thực hiện được điều này, ngoài địa chỉ IP, một thông số khác gọi là **Subnet Mask** cần được xác định cho máy. Subnet Mask gồm 4 số nguyên không dấu, mỗi số gồm 8 bit. Giá trị của subnet mask gồm 32 bit được chia làm hai phần: phần bên trái gồm những bit 1 và phần bên phải gồm những bit 0. **Các bit 0 xác định những địa chỉ IP nào cùng nằm trên một mạng con với nó**. Ví dụ như, thông thường subnet mask của một địa chỉ IP trong lớp C là 255.255.255.0.

Khi cần gởi thông tin ra ngoài mạng con, các máy cần phải biết địa chỉ IP của các **máy trung gian (Gateway)**.

Do địa chỉ IP chỉ là những con số không có tính gợi nhớ nên trong Internet người ta thường sử dụng thêm một **dịch vụ định tên miền** (**Domain Name Service)** cho các máy sử dụng trong mạng. Mỗi máy sử dụng trong mạng có thể được gán cho một hoặc nhiều tên khác nhau. Dạng của Domain Name: **host.subdomain.domain**

- **host** : là tên máy
- **subdomain** : chỉ ra một tổ chức mạng nhỏ hơn trong domain
- **domain**: định danh cho tên một tổ chức mạng lớn như các công ty đa quốc gia, các quốc gia, …

Ví dụ: aao.hcmut.edu.vn || server.empac.com

Khi một máy X muốn gởi thông tin đến máy có domain name là A, máy X cần phải tìm ra địa chỉ IP thật sự của máy A. **Máy định tên miền (Domain Name Server – DNS)** xác định giùm địa chỉ IP thật của máy A. Khi DNS của ISP không giải quyết được việc chuyển đổi cho Client, yêu cầu giải quyết sẽ được chuyển lên DNS cấp cao hơn giải quyết.

Do có nhiều loại dịch vụ trên Internet cùng sử dụng chung Internet Protocol, ngoài địa chỉ IP, người ta đưa thêm vào khái niệm **cổng (port)**. Mỗi loại dịch vụ sẽ sử dụng một port khác nhau để hoạt động.

Port là một số nguyên dương 16 bit nên có giá trị 1→ 65535, trong đó :

- **1→ 999** : các port dành cho các ứng dụng mạng phổ biến như :
  - port 80 : ứng dụng Web
  - port 25 : ứng dụng gởi mail
  - port 110 : ứng dụng trả mail
- **1000 – 65535** : các port do chúng ta định nghĩa cho ứng dụng của mình

Một số dịch vụ phổ biến trên Internet:

- **Dịch vụ World Wide Web (WWW):** Dịch vụ này đưa ra cách truy xuất các tài liệu của các máy trên mạng dễ dàng qua các giao tiếp đồ họa. Các tài liệu này liên kết với nhau tạo nên "kho" tài liệu khổng lồ. Thông qua Internet, các Browser như Microsoft Edge, Chrome hay Firefox truy nhập được thông tin của Web Server bằng **URL (Uniform Resource Locator)**: "Method:// Host.Subdomain.Domain:Port"

- **Method**: cho biết loại dịch vụ được sử dụng. Ví dụ như **http** cho web, **ftp** để truy xuất file, ...
- **Host.Subdomain.Domain**: cho biết tên của Server cần truy xuất. Có thể sử dụng IP của Server.
- Thông thường các port được định nghĩa sẵn nên không cần ghi :Port trong URL. Ví dụ : http://www.empacbk.com mà không cần ghi port 80. Khi lập trình Web có xài các Web Server như Tomcat chẳng hạn thì ta phải thêm phần :8080 trong URL.

- **Dịch vụ E-mail**: Dịch vụ này cho phép các cá nhân trao đổi thư với nhau thông qua Internet. Để sử dụng dịch vụ này người sử dụng cần mở một **hộp thư (Mailbox)** tại các máy ISP hoặc trên internet như Gmail, Yandex. Sau khi mở hộp thư, người sử dụng được cấp một địa chỉ E-mail và một mật mã (password) để truy xuất hộp thư của mình. Người dùng có thể xài một chương trình **Mail Client** để truyền nhận thư của mình từ hộp thư trên máy Server như **OutLook Express** chẳng hạn, hoặc xài Web Mail Client như Gmail Web.

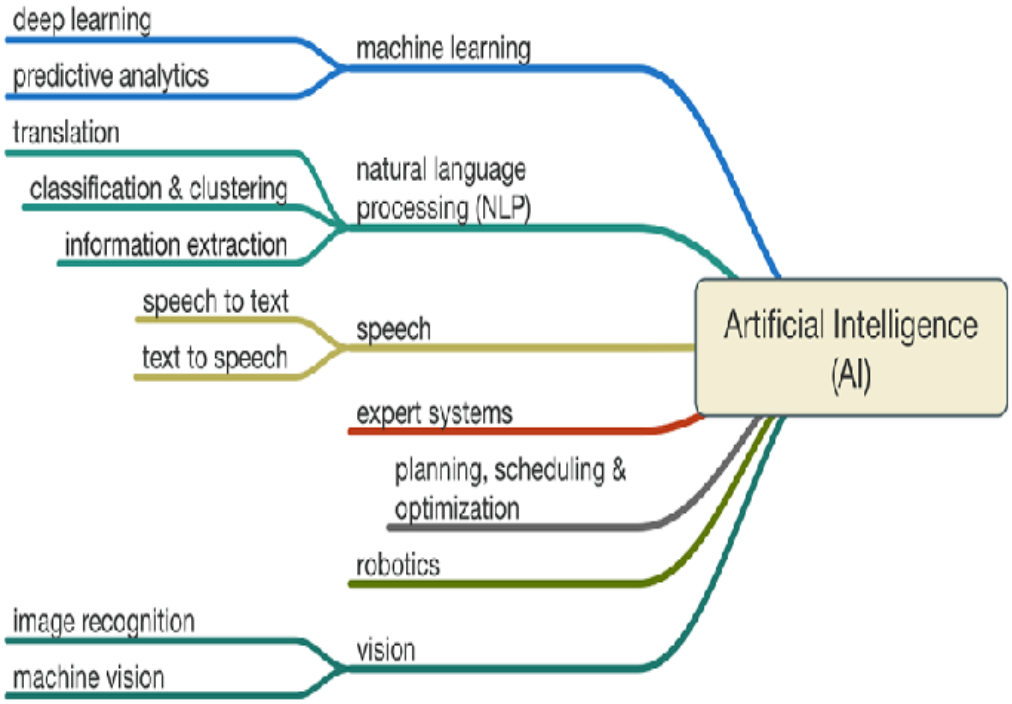
  Chương trình quản lý các hộp thư tại máy Server gọi là **Mail Server**.

  Địa chỉ của một hộp thư khi truyền nhận thư qua Internet được gọi là Internet E-Mail Address có dạng: mailbox@mailserver.subdomain.domain, ví dụ: homertruong66@gmail.com.
- **Dịch vụ FTP**: Là dịch vụ truyền/nhận tập tin trên Internet. Thông qua dịch vụ này Client có thể download các tập tin từ Server về máy cục bộ hay upload các tập tin vào Server. Dịch vụ này rất thường được sử dụng để sao chép các phần mềm freeware, shareware, các bản update cho driver ... Tên của các FTP Server thường có dạng: ftp.subdomain.domain (VD : ftp.empacbk.com)

Các điểm bạn cần nắm vững trong mảng Mạng máy tính

1. Tại sao MMT xuất hiện.
2. Các loại MMT (xét theo qui mô).
3. Các giao thức phổ biến của MMT.
4. Mô hình MMT lý thuyết (OSI) và thực tế (TCP/IP).
5. Cách định danh các MVT trong mạng.
6. Tên miền & dịch vụ xác định tên miền.
7. Các dịch vụ phổ biến trên Internet.

5) *Trí tuệ nhân tạo (Artificial Intelligence)*



Trí tuệ nhân tạo là lĩnh vực trong đó con người giúp MVT có trí tuệ như con người để tận dụng sức mạnh của MVT xử lý những vấn đề mà con người không xử lý nổi do giới hạn về thời gian hoặc năng lực tính toán. Một số ứng dụng phổ biến của AI:

- Người máy (Robot)
- Nhận dạng khuôn mặt (Face Recognition)
- Nhận dạng giọng nói (Speech Recognition)
- Dự báo thời tiết (Weather Forecast)

Công thức cơ bản của AI rất đơn giản: **Mô hình (Presentation)** & **Tìm kiếm (Search)**.
Điều này có nghĩa là bất cứ bài toán nào của AI thì bắt đầu với việc con người mô hình hóa các kiến thức mà họ đang có để máy học, sau đó máy sẽ tìm kiếm các giải pháp cho vấn đề của con người bằng các thao tác tìm kiếm các giải pháp trên mô hình trên. Ví dụ: trong trò chơi cờ tướng, con người thể hiện bàn cờ với các quân cờ, sau mỗi nước đi của người thì máy sẽ tìm nước đi phù hợp dựa trên trạng thái (hình vị trí các quân cờ) của bàn cờ.

Các điểm bạn cần nắm vững trong mảng Trí tuệ nhân tạo.

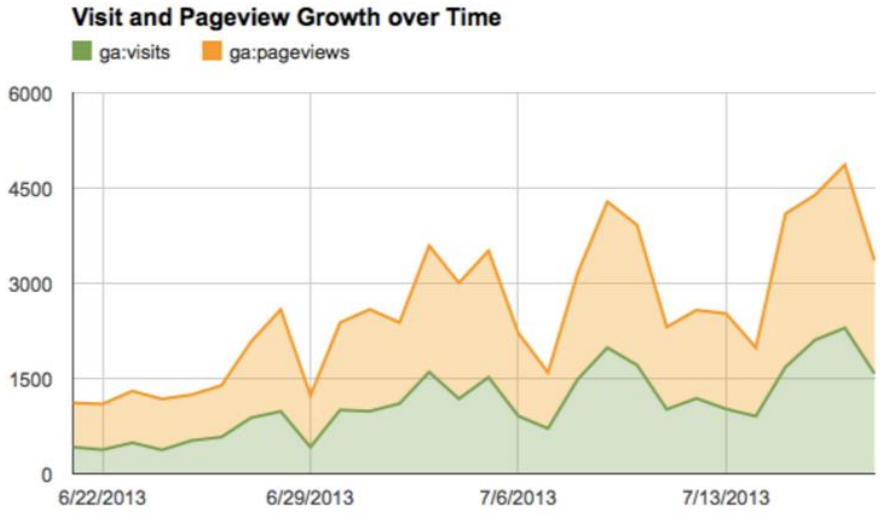
1. Trí tuệ nhân tạo là gì.
2. Các ứng dụng phổ biến của trí tuệ nhân tạo.
3. Một số giải thuật phổ biến của AI, gọi là Machine Learning Algorithms.

6) Khoa học dữ liệu (Data Science)

a) Xử lý dữ liệu (Data Analytics)

Data Analytics là kỹ thuật phân tích dữ liệu của **khách hàng (Customer)** để tìm hiểu **hành vi (Behavior)** và **xu hướng (Trend)** của họ, từ đó giúp cho **doanh nghiệp (Enterprise/Business)** đưa ra giải pháp để tăng **doanh thu (Revenue)**. Vì thế đây là kỹ thuật được dùng nhiều trong mô hình **B2C (Business-2-Customer)**.

Ví dụ: Google Analytics là một công cụ để theo dõi lượng khách tham quan 1 website và xem các trang web của website đó.



Một ví dụ khác là trên 1 trang web thương mại điện tử, ta có thể theo dõi những mặt hàng mà khách hàng hay tương tác mà ko mua (bỏ vô/ra giỏ hàng nhưng không lấy) để giảm giá/khuyến mãi một chút nhằm kích thích khách hàng mua.

b) Khai thác dữ liệu (Data Mining)

(Phần này có khá nhiều thuật ngữ chuyên ngành nên tôi không dịch ra tiếng Việt, bạn nào đi vào ngành này sẽ tìm hiểu kỹ hơn.).

Data mining is the extraction of implicit, previously unknown, and potentially useful information from data. Data mining is defined as the process of discovering patterns in data. The process is preferably fully automatic, but it is often semi-automatic due to performance. The patterns discovered must be meaningful in that they lead to some advantage, usually an economic advantage.

Machine learning provides the technical basis of data mining. Machine learning is concerned with the design and development of algorithms and techniques that allow computers to "learn". Machine learning has a large number of applications including natural language processing, speech and speaker recognition, pattern classification, to name a few.

Generally, machine learning styles in data mining include:

- **Numeric prediction**: predicts a target value based on a vector of features.
- **Pattern classification**: learns a way of classifying unseen patterns into discrete classes from a set of labeled examples.
- **Association learning**: any association among features is sought, not just ones that predict a particular class value.
- **Clustering**: seeks groups of samples that belong together.

In these four styles, numeric prediction and pattern classification styles are used more widely than the other two in data mining applications.

Numeric Prediction

Given a column feature vector $X = (X_1, X_2,\ldots, X_d)^T$, where $X_1, X_2, \ldots, X_d$ are d features of pattern X and d is the number of dimensions of the feature vector X.

The problem is to predict the numeric value $Y = f(X)$, where f(X) is a function with respect to X. For example: $f(X) = X_1 + 2X_2$ or $f(X) = 3X_1 – X_2$.

Pattern Classification

Given a column feature vector $X = (X_1, X_2,\ldots, X_d)^T$, where $X_1, X_2, \ldots, X_d$ are d features of pattern X and d is the number of dimensions of the feature vector X.

The problem is to classify X into one of the k classes $\{C_1, C_2,\ldots, C_k\}$.

To solve numeric prediction and pattern classification problems simply, we can use one of the three basic learning models:

- Zero-Rule Model → CART
- Linear Regression Model → Model Tree
- Logistic Regression Model → Logistic Model Tree

It is seldom known in advance which procedure will perform best or even well for any given problem, so the **"trial-and-error" approach** is always employed in practical data mining application. Particularly, it is tempting to try out different learning schemes with different combinations of their options on a given dataset to select the one that works best.

Among numerous of learning schemes, Decision Tree has been the most widely used algorithm in practice because it has the following advantages and disadvantage:

- Decision Tree is interpreted so easily because it uses subset of attributes. Practical data mining applications generally require interpretable models.
- Decision Tree efficiently classifies new samples by simply traversing the tree structure without requiring much computation**.**
- Decision Tree can be applied to any kind of data structure: mixed data type (nominal and numeric data) and data with high dimensionality.
- Decision Tree helps to determine which attribute is the most important for numeric prediction and pattern classification.
- Decision Tree is appropriate for limited dataset.
- It is difficult to design an optimal tree, probably leading to a large tree with poor error rates.

Therefore, decision tree is usually chosen to be the based learning scheme in most data mining applications.

A practical example with Decision Tree

Let us consider a practical example about a golf club:

"A manager of a golf club wants to predict customer attendance in his club in order to know if he should hire extra staffs on days when customers play golf or dismiss most of them on days when customers do not play golf. "

The manager observed and measured four features in two weeks:

- **Outlook**: a nominal feature that has value belonging to a set of possible values (sunny, forecast, rain);
- **Temperature**: a numeric feature. The unit of measurement is ^{o}C;
- **Humidity**: a numeric feature. The unit of measurement is %;
- **Wind**: a nominal feature that has value belonging to a set of possible values (true, false).

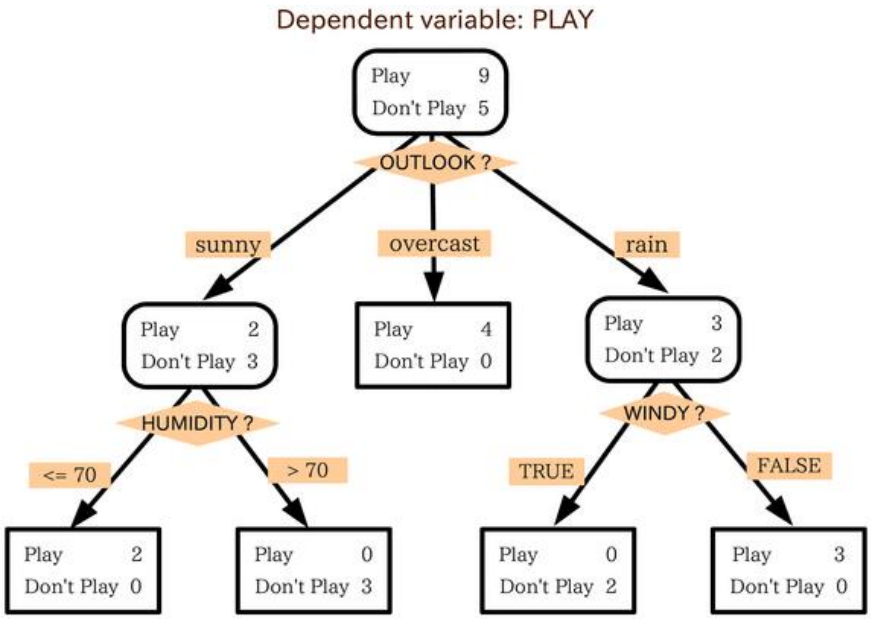
He wants to know if the customers will play or not in a given day, so the target value is a category class belonging to a set of classes: {Play, Don't play}.

Then, he made a dataset:

Play golf dataset

Independent variables				Dep. var
OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
sunny	85	85	FALSE	Don't Play
sunny	80	90	TRUE	Don't Play
overcast	83	78	FALSE	Play
rain	70	96	FALSE	Play
rain	68	80	FALSE	Play
rain	65	70	TRUE	Don't Play
overcast	64	65	TRUE	Play
sunny	72	95	FALSE	Don't Play
sunny	69	70	FALSE	Play
rain	75	80	FALSE	Play
sunny	75	70	TRUE	Play
overcast	72	90	TRUE	Play
overcast	81	75	FALSE	Play
rain	71	80	TRUE	Don't Play

The decision tree built from this dataset is as follow:



According to this decision tree, he can conclude that:

- Users play golf on sunny and non-humid days / on overcast days / or on rainy and non-windy days, then he will hire extra staffs on these days.
- Users don't play golf on sunny and humid days, or on rainy and windy days, then he will dismiss most of the staff on these days.

c) Kho dữ liệu (Data Warehouse)

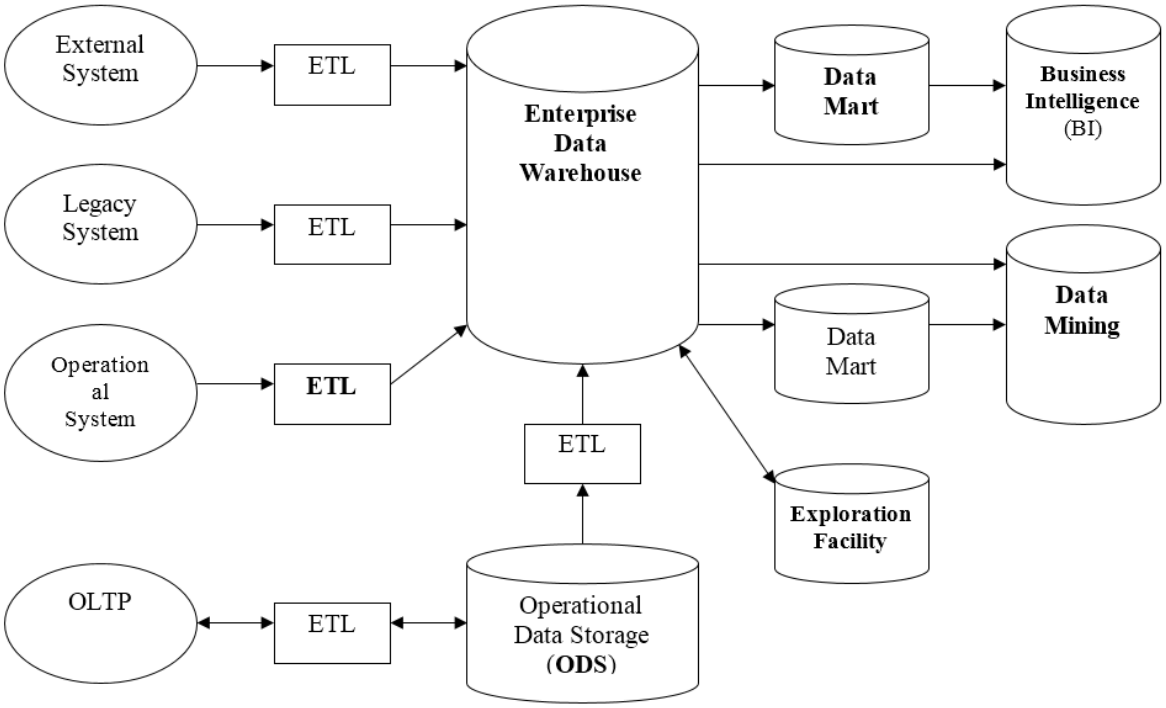
Trong một doanh nghiệp lớn có nhiều phòng ban, cơ quan khác nhau. Mỗi nơi xài những phần mềm quản lý với các CSDL khác nhau. Điều này dẫn đến việc chia sẻ thông tin giữa các nơi trở nên hơi khó khăn hơn. Vì vậy kho dữ liệu được thiết kế để giải quyết vấn đề này.

Kho dữ liệu của doanh nghiệp chứa dữ liệu từ nhiều nguồn khác nhau như từ các phòng ban hoặc online, bao gồm cả **dữ liệu có cấu trúc (structured data)** và **dữ liệu không cấu trúc (unstructured data)**. Dữ liệu từ nguồn nào đó vô kho dữ liệu qua quá trình **Trích xuất-Chuyển đổi-Nạp (Extract-Transfer-Load – ETL)**.

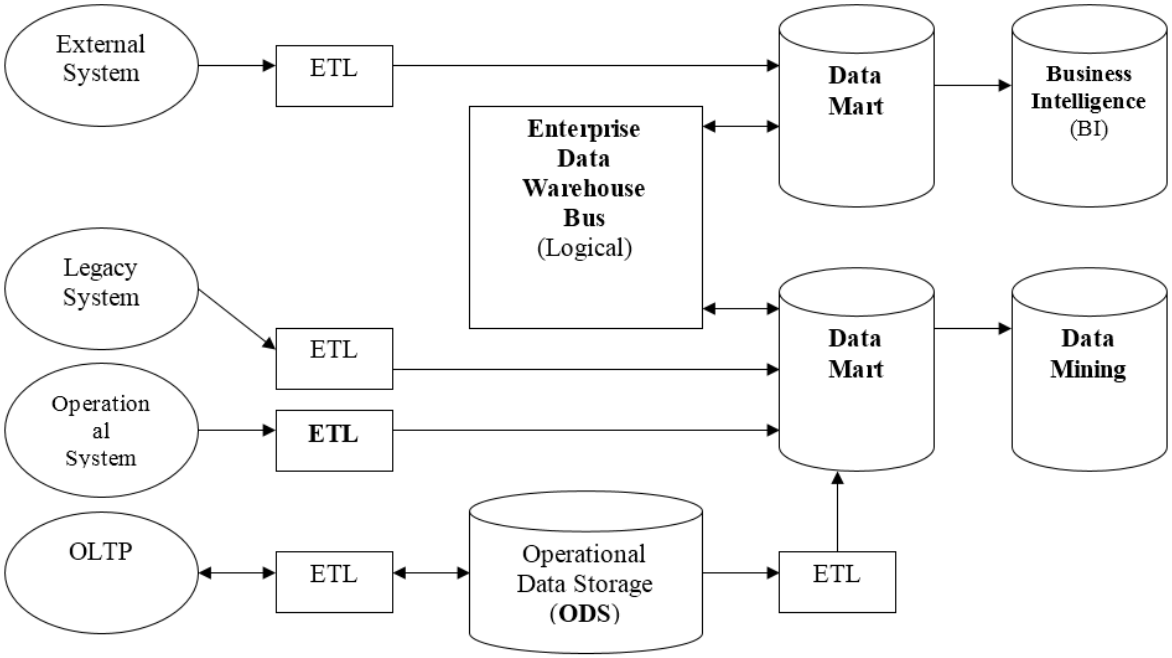
Kho dữ liệu có thể được xây dựng theo 2 cách:

- **Từ trên xuống (Top-down) theo William Innmon**
  - Theo cách thiết kế này thì kho dữ liệu được xây dựng từ lúc doanh nghiệp bắt đầu hoạt động. Dữ liệu hỗn tạp từ các hệ thống ngoài doanh nghiệp, hệ thống cũ, hệ thống vận hành và các **xử lý giao dịch online (OnLine Transaction Processing – OLTP)** đi qua quá trình ETL đưa ra dữ liệu đồng nhất đổ vào **Kho dữ liệu doanh nghiệp (Enterprise Data Warehouse)**.
  - Dữ liệu từ kho dữ liệu sau đó sẽ cung cấp dữ liệu khác nhau cho các **buồng dữ liệu (Data Mart)** khác nhau của các phòng/ban khác nhau nhằm phục vụ cho các công cụ của doanh nghiệp như **Phân tích chiến lược(Business Intelligence – BI)**, **Khai thác dữ liệu (Data Mining)**.
- **Từ dưới lên (Bottom-up) theo Ralph Kimball**
  - Theo cách thiết kế này thì kho dữ liệu được xây dựng sau khi doanh nghiệp đã hoạt động. Dữ liệu hỗn tạp từ các hệ thống ngoài doanh nghiệp, hệ thống cũ, hệ thống vận hành và các xử lý giao dịch online (OnLine Transaction Processing – OLTP) đi qua quá trình ETL đổ dữ liệu vào các buồng dữ liệu (Data Mart) khác nhau của các phòng/ban khác nhau nhằm phục vụ cho các công cụ của doanh nghiệp như Phân tích chiến lược(Business Intelligence – BI), Khai thác dữ liệu (Data Mining).
  - Kho dữ liệu doanh nghiệp **luận lý (logical)** được tạo ra để các **buồng dữ liệu (Data Mart)** khác nhau của các phòng/ban khác nhau có thể truy xuất/trao đổi dữ liệu lẫn nhau khi cần.

Top-down Data Warehouse



Bottom-up Data Warehouse



Các điểm bạn cần nắm vững trong mảng Khoa học dữ liệu:

1. Khoa học dữ liệu là gì.
2. Các nhánh của khoa học dữ liệu và các ứng dụng phổ biến của nó.

II. Các nghề IT phổ biến

Trước khi tìm hiểu các nghề IT phổ biến, chúng ta cần quan sát xem ngoài thị trường có các loại công ty IT kiểu nào, mô hình hoạt động của chúng ra sao.

Thực tế hiện nay có 3 loại công ty IT phổ biến:

- **Công ty vừa và nhỏ (Small and Medium Enterprises – SME)**: Đây là loại công ty xây dựng các giải pháp IT, thị trường đã có, rồi đem bán cho một/vài thị trường ngách.
- **Công ty gia công IT (IT Outsourcing Companies – ITOC)**: Đây là loại công ty nhận yêu cầu từ khách hàng và bảo trì/xây dựng một phần/toàn bộ sản phẩm IT cho họ.
- **Công ty khởi nghiệp công nghệ (Tech Startups - TS)**: Đây là loại công ty có ý tưởng xây dựng một sản phẩm mà thị trường chưa có/có rất ít, mang tính đột phá cao, có thể thay đổi thế giới như Facebook, Uber, Grab, Airbnb…

Bất kể loại hình kinh doanh thế nào, cả 3 loại công ty trên đều xây dựng sản phẩm là **hệ thống IT (IT System)** phục vụ cho **chiến lược kinh doanh (Business Strategy)** của các doanh nghiệp.

- Một hệ thống IT có thể gồm một/nhiều phần mềm được phát triển bởi một/nhiều **nhóm phát triển (Development Team).**
- Trong một nhóm phát triển có nhiều **vai trò (Role)**.
- Mỗi vai trò có thể được đảm trách bởi một/nhiều **nhân viên (Employee)**.
- Mỗi nhân viên có thể nhận một/nhiều vai trò ở một thời điểm nào đó.
- Mỗi nhân viên chỉ có 1 **nghề (Job)** ở một thời điểm.
- Khi nói đến làm việc nhóm thì luôn cần có 1 qui trình để mọi người trong nhóm cùng theo để giúp việc phối hợp trong nhóm đạt hiệu quả cao nhất. Qui trình xây dựng hệ thống IT đang được dùng nhiều nhất hiện nay là SCRUM. Có các vai trò sau trong 1 SCRUM team:
  - **Chủ sản phẩm (Product Owner – PO)**
  - **Trưởng nhóm Marketing (Marketing Lead)**
  - **Nhà phân tích nghiệp vụ (Business Analyst – BA)**
  - **Trưởng nhóm (Team Lead – TL)**
  - **Nhà phát triển phần mềm (Software Developer – SD)**
  - **Kiểm tra chất lượng sản phẩm (Quality Control – QC)**
  - **Kiểm tra chất lượng qui trình (Quality Assurance – QA)**
  - **Nhà quản trị CSDL (Database Admin – DBA)**
  - **Nhà khoa học dữ liệu (Data Scientist)**
  - **Chuyên viên triển khai vận hành hệ thống (DevOps)**
  - **Nhà quản lý dự án (Project Manager – PM)**

Ví dụ: 1 team A có 1 anh X vừa làm Role SD vừa làm DevOps; 1 team B có 2 chị Y1, Y2 cùng làm BA.

Trong suốt 15 năm làm việc, ngoại trừ vai trò QA, Marketing Lead, tôi đã trải qua hết các vai trò còn lại trong các SCRUM Teams tôi từng tham gia.
Lý do mà tôi đã làm qua khá nhiều vai trò như thế là bởi vì các công ty tôi tham gia phần lớn là SME & TSs. Mà các công ty này thì nguồn nhân lực thời gian đầu hạn chế, cho nên các thành viên trong SCRUM Team phải kiêm nhiều vai trò.

Sau đây tôi sẽ chia sẻ các nghề IT phổ biến mà bạn có thể định hướng cho mình hướng tới trong lúc học IT hoặc sau khi học xong IT.

Cho dù bạn chọn nghề IT nào trong số các nghề tôi chia sẻ thì bạn cũng nên biết các nghề IT khác có liên quan là gì, vì cũng có lúc bạn phải tạm thời kiêm luôn các nghề IT đó khi công ty thiếu người như tôi đã từng trải qua.

Ngoài ra việc hiểu các nghề IT/vai trò liên quan giúp bạn phối hợp/hỗ trợ tốt hơn với các thành viên khác trong SCRUM team.

1) Hỗ trợ kỹ thuật IT (IT Support)

IT Support trong một công ty có nhiệm vụ hỗ trợ tất cả nhân viên của công ty (đôi khi gồm cả khách hàng bên ngoài) các vấn đề liên quan đến:

- Thiết bị máy móc (Devices, Machines): lắp đặt, cấu hình
- Phần cứng MVT: cài đặt, sửa chửa
- Phần mềm MVT: Email, Chat…
- MMT: cấu hình, chỉnh sửa
- Các vấn đề về **bảo mật (Security)**: cấp **tài khoản (Account)**, chống **tin tặc (Hacker)**

Khi 1 nhân viên mới vào công ty, IT Support sẽ cung cấp MVT, cài đặt tài khoản công ty. Trong quá trình làm việc, nhân viên có thể có nhu cầu nâng cấp RAM, có nhu cầu cần 1 **máy chủ chạy demo (stage server)** để triển khai phần mềm cho **BA** kiểm tra, hoặc gặp vấn đề kết nối với mạng của khách hàng… thì IT Support sẽ hỗ trợ. Khi 1 nhân viên nghỉ công ty, IT Support sẽ xóa tài khoản của họ, thu hồi những thiết bị/tài nguyên đã cung cấp cho họ.

Để làm IT Support tốt thì cần có những đặc điểm sau:

Tính tình	• Thích hỗ trợ người khác • Cẩn thận • Trách nhiệm
Kiến thức	• Tổng quan về các thiết bị phần cứng: MVT, màn hình, máy in… • Kiến thức chung về HĐH: Linux, Windows… • Kiến thức chung về MMT
Kỹ năng	• Anh Văn (English) • Suy nghĩ luận lý (Logical Thinking) • Giải quyết vấn đề (Problem Solving)

2) *Phân tích nghiệp vụ (Business Analyst – BA)*

BA là người có nhiệm vụ phân tích nghiệp vụ của một phần mềm:

- **Các luồng nghiệp vụ (Business Flows)**
- **Các qui luật nghiệp vụ (Business Rules)**
- **Các tính năng (Features)**
- **Các tương tác người dùng (User Stories)**

Ví dụ: một phần mềm quản lý các sự kiện cho sinh viên (SV) tham gia.

- Một biz flow về SV tham gia sự kiện:
  - 1 nhà tổ chức sự kiện tạo 1 sự kiện trên hệ thống
  - 1 SV vào đăng ký tham gia sự kiện đó
  - Khi SV đến tham dự sự kiện thì nhà tổ chức check-in SV đó
- Một biz rule là SV đó có đủ điều kiện tham gia sự kiện đó ko? (tuổi, trường, khu vực…)
- Tính năng này có thể được đặt là Quản lý sự kiện (Event Management)
- Một tương tác ví dụ là "SV đăng ký tham gia 1 sự kiện"
  - SV vào trang chủ hệ thống
  - SV thấy 1 sự kiện
  - SV bấm vào xem chi tiết
  - SV nhất nút đăng ký tham gia sự kiện

Trước đây, một SD khi thực hiện một tính năng của phần mềm thì sẽ phân tích, thiết kế rồi lập trình để thực thi tính năng đó. Như vậy, SD đã làm luôn vai trò BA. Sau này khi phần mềm càng ngày càng lớn hơn và phức tạp hơn nhiều thì SD sẽ trở nên quá nhiều việc.

Chính vì vậy nghề BA ra đời để giảm tải cho SD nhằm giúp SD tập trung vô nhiệm vụ chính là thiết kế, hiện thực phần mềm, còn mảng nghiệp vụ phần mềm để BA lo.

Tùy theo loại hình công ty mà BA sẽ có công việc khác nhau tí:

- Trong **ITOC** thì BA là cầu nối giữa SCRUM Team của công ty và **chủ sản phẩm phần mềm (Product Owner – PO)** phía khách hàng.
- Trong **SMEs/TSs** thì BA là cầu nối giữa **nhóm phát triển phần mềm (Software Development Team)** và PO của công ty.

Theo qui trình SCRUM, BA sẽ làm những việc sau:

- Giai đoạn chuẩn bị dự án:
  - ✓ Thu thập yêu cầu phần mềm từ PO
  - ✓ Phân tích yêu cầu để xác định Business Flows
  - ✓ Xác định Business Rules
  - ✓ Định nghĩa Features
  - ✓ Với mỗi Feature, xác định các User Stories

- Giai đoạn dự án đang diễn ra
  - ✓ Làm việc với PO để xác định các User Stories/Tasks ưu tiên cần làm trước mỗi **Sprint**
  - ✓ Đặc tả User Stories/Tasks trong Sprint
  - ✓ Kiểm tra lại các User Stories mà SD làm xong đưa lên **Stage Server** có đúng yêu cầu hay không
  - ✓ Khi **xuất các tính năng sản phẩm (Release)** lên **Production Server** thì kiểm tra chất lượng xem đúng yêu cầu không
  - ✓ Kiểm tra phần tích hợp với các Releases trước: **Kiểm tra tích hợp (Integration Validation)**
  - ✓ Làm việc với SD để giải quyết các vấn đề (nếu có): **Lỗi (Bugs), Cải tiến hệ thống (System Improvement)**.
  - ✓ Thông báo Bugs hoặc Improvement (nếu có) cho SCRUM Team.

- Giai đoạn dự án sắp hoàn thành
  - ✓ Kiểm tra lại toàn bộ hệ thống (System Validation)

Để làm BA tốt thì cần có những đặc điểm sau:

Tính tình	• Thích phân tích nghiệp vụ • Thích làm cầu nối giữa các nhóm chức năng khác nhau • Có trách nhiệm

Kiến thức	• Kiến thức tổng quan về **công nghệ phần mềm (CNPM)**
	• Kiến thức nghiệp vụ càng nhiều ngành càng tốt, ví dụ:
	  ○ **Quản lý doanh nghiệp (Enterprise Resource Planning – ERP)**
	  ○ **Thương mại điện tử (e-Commerce)**
	  ○ **Chăm sóc sức khỏe (Health Care)**
	  ○ **Tiếp thị số (Digital Marketing)**…

Kỹ năng	• Anh Văn (English) • Suy nghĩ luận lý (Logical Thinking) • Giải quyết vấn đề (Problem Solving) • Phân tích nghiệp vụ (Business Analysis) • Giao tiếp tốt (Good Communication) • Làm việc nhóm (Teamwork)

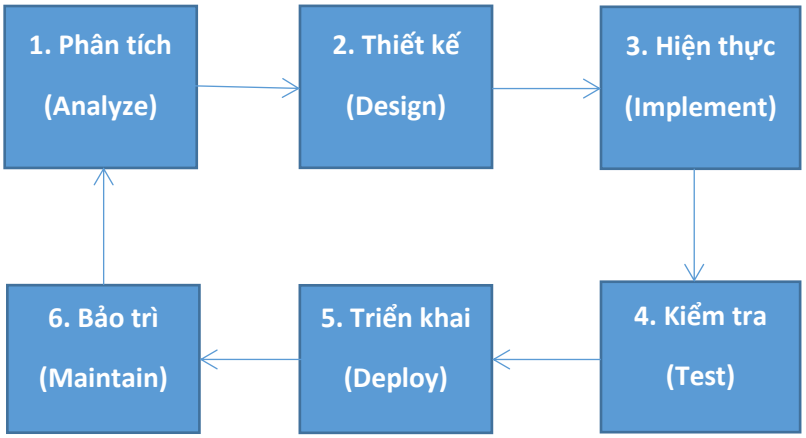
Bạn không cần kiến thức IT từ đầu mà vẫn có thể theo nghề BA, cứ làm rồi từ từ học thêm.

Các nấc thang mà bạn phải leo để lên đến đỉnh cao của nghề BA (thời gian leo ở mỗi nấc tùy theo nỗ lực và sự kiên trì của bạn).

1 Fresher BA	Bạn mới vào nghề BA, thường bạn sẽ tham gia vô 1 dự án đang chạy để hỗ trợ các senior BA. Nhiệm vụ của bạn lúc này là học hỏi để nắm rõ hết nghiệp vụ của tất cả dự án bạn tham gia, và học luôn kiến thức/kinh nghiệm của các senior BA.
2 BA	Ở mức này bạn đã có kiến thức về nghiệp vụ của khá nhiều lĩnh vực và có kinh nghiệm BA tương đối vững để có thể tham gia 1 dự án từ đầu. Nhiệm vụ của bạn là tích lũy thêm kiến thức lĩnh vực và kinh nghiệm BA.
3 Senior BA	Ở mức này bạn đã có kiến thức về nghiệp vụ và có kinh nghiệm BA đầy mình. Nhiệm vụ của bạn là vẫn tích lũy thêm kiến thức lĩnh vực và biết cách phân chia việc cho các BA khác phụ mình (nếu cần).

3) Phát triển phần mềm (Software Developer – SD)

Trong số các nghề IT trong SCRUM Team thì SD là nghề phổ biến và có nhu cầu cao nhất. Về cơ bản, một SD là người đóng góp và việc xây dựng một phần mềm với qui trình 6 bước lặp được gọi là **CNPM**:



Thông thường:

- Phân Tích được thực hiện bởi BA
- SD làm Thiết Kế & Hiện Thực.
- Kiểm Tra được làm bởi Quality Control (QC)
- Triển Khai do DevOps đảm trách
- Cả SCRUM Team làm Bảo trì.

Qui trình này cũng chính là **vòng đời phát triển phần mềm (Software Development Life Cycle).**

Đôi khi SD cũng giúp BA xem xét các nghiệp vụ phức tạp và giúp QC kiểm tra những hành xử của phần mềm đôi khi khá mập mờ khó hiểu do người dùng tương tác theo 1 trình tự ngẫu nhiên nào đó.

Khi nói đến Thiết Kế, có nhiều loại Thiết Kế trong đó Thiết Kế **Kiến trúc phần mềm (Software Architecture)** là phần quan trọng nhất và cũng là khó nhất. Phần này được đảm trách bởi 1 SD có kinh nghiệm gọi là **Kiến trúc sư phần mềm (Software Architect – SA)**, người mà có kiến thức và kinh nghiệm khá mạnh trên nền tảng được chọn để xây dựng phần mềm, ví dụ: Java, .Net, PHP, Ruby platforms… Hiển nhiên người này sẽ nắm vai trò **trưởng nhóm (Team Lead)**.

Một cách trực quan, chúng ta có thể định nghĩa ra 3 mức của SD dựa trên kinh nghiệm:

- **Software Developer (SD)**: là người mới ra trường hoặc mới đi làm ít kinh nghiệm
- **Senior SD (SSD)**: là SD có nhiều kinh nghiệm về công nghệ và kiến thức trên một số lĩnh vực họ đã làm qua.
- **Software Architect (SA)**: là SSD có nhiều kinh nghiệm để có thể Thiết Kế và Hiện Thực kiến trúc phần mềm.

Trong các công ty gia công phần mềm thì số lượng mức có thể nhiều hơn, ví dụ:

- Fresher
- Associate Software Developer (ASD)
- Software Developer (SD)
- Senior Software Developer (SSD)
- Principle Software Developer (PSD)
- Software Architect (SA)
- Senior Software Architect (SSA)

Ở mỗi mức có thể chia theo những mức con nữa, ví dụ: SD L1, SSD L3, PSD L2… Sở dĩ có nhiều mức thế này là để các công ty có thể thành lập các nhóm khác nhau với các SDs ở các mức khác nhau nhằm đáp ứng nhu cầu các dự án khác nhau của khách hàng.

Khi tham gia một nhóm phát triển phần mềm tương đối lớn:

- PSD, SA & SSA sẽ lập ra 1 nhóm gọi là Hội đồng đánh giá kiến trúc (**Architecture Review Board – ARB**). Nhóm này có nhiệm vụ xem xét tất cả những thiết kế làm thay đổi kiến trúc phần mềm.
- ASD, SD & SSD sẽ tham gia phần Thiết kế & Hiện thực phần mềm.

Làm SA cho một dự án phần mềm mới hoàn toàn khó hơn rất rất nhiều so với một hệ thống IT đã có sẵn.

Trong các SMEs/TSs, các nhà **đồng sáng lập (Co-Founders)** có thể sẽ phải làm việc với vai trò BA, SA và SD vì nguồn tài chính hạn chế.

Với sự xuất hiện của kiểu kiến trúc phần mềm **REST**, **Micro-Services Architecture (MSA)**, **công nghệ Frontend (FE)** để xây dựng các **ứng dụng 1 trang (Single Page App – SPA)** và **lập trình ứng dụng di động (Mobile Programming)**, SD có thể chọn làm việc trên các phần sau đây:

- **Frontend (FE)**:
  - **Web**: Javascript frameworks như JQuery, React, Redux, Vue, Angular, Ember…
  - **Mobile**: iOS, Android frameworks
- **Backend (BE)**: Java, .Net, PHP, Python, Ruby frameworks…

Một SD làm việc trên cả FE, BE gọi là **Full-stack Developer**.

Một SD có thể làm việc trên nhiều nền tảng khác nhau (Java, .Net…) gọi là **Cross-functional Developer**.

Trên thị trường việc làm, nhiều tên cho các vị trí tuyển SD được sử dụng, sau đây là vài tên phổ biến:

- Software Developer
- Software Engineer
- Programmer
- Software Architect

Ngoài ra, các nhà tuyển dụng ngày nay thường xài các thuật ngữ công nghệ chi tiết để thêm vô tên, chẳng hạn: Java Developer, .Net Software Architect, Ruby Technical Architect, Frontend Programmer, Java Backend Developer, iOS Developer, NodeJS Developer…

Bản thân tôi cũng đã từng được đặt cho một số vị trí trong các công ty phần mềm: Software Engineer, Programmer Analyst, Computer Scientist, Principle Software Engineer, Software Architect.

Là một SD rất là thú vị và cũng đầy thách thức.

- Phần thú vị là chúng ta có thể vừa làm vừa thưởng thức một tách cà phê sữa để xây dựng những phần mềm hữu ích cho xã hội nhằm tăng năng suất làm việc cũng như tiết kiệm thời gian và tiền bạc. Đôi khi chúng ta đụng phải một sự cố kỹ thuật, chúng ta có khuynh hướng bỏ ra hàng giờ để giải quyết nó. Nhiều lúc chúng ta có thể xử lý nhanh, đôi lúc chúng ta cũng đầu hàng sau hàng giờ cố gắng nghiên cứu. Đối với các vấn đề phức tạp, chúng ta mất vài ngày để giải quyết là chuyện bình thường. Thỉnh thoảng chúng ta có khả năng tìm ra giải pháp cho một vấn đề cực kỳ hóc búa khi chúng ta đang hát và tắm dưới vòi sen trong khi chúng ta đã nghĩ chúng ta không thể làm được sau rất nhiều lần thử.
- Các thách thức mà một SD gặp phải mỗi ngày rất nhiều. Vấn đề nhức đầu nhất của SD là ác mộng làm thêm ngoài giờ (Over Time – OT) bởi vì ước tính thiếu (Under Estimation – ETA). Thông thường một SD làm việc 8 giờ mỗi ngày. Nếu họ không thể hoàn tất công việc đúng hạn thì họ phải bỏ thêm thời gian ngoài giờ để làm tiếp. Ác mộng làm thêm ngoài giờ hay xảy ra với những ai ít hoặc chưa có kinh nghiệm và hầu hết trường hợp là chưa nắm vững CNPM. Vì thế, nắm vững CNPM là chìa khóa để giảm/bỏ cái vấn đề phổ biến không hề mong muốn này.

Ngoài ác mộng làm thêm ngoài giờ, SD cũng có các vấn đề khác sau đây:

- Không có một tầm nhìn rộng bao quát thế giới IT bởi vì khi học ở trường hay đi làm cũng chỉ biết những chủ đề cụ thể, không ai chỉ cho kiến thức/kinh nghiệm tổng hợp của các nhánh trong ngành IT
- Gặp khó khăn lớn để hiểu một hệ thống IT một cách đầy đủ
- Mất rất nhiều thời gian để học một công nghệ mới vì không có nền tảng IT vững chắc
- Mơ hồ về hướng phát triển sự nghiệp

Để làm SD tốt thì cần có những đặc điểm sau:

Tính tình	• Biết suy luận • Luôn học hỏi • Có trách nhiệm
Kiến thức	• Lập trình hàm và hướng đối tượng (Functional and OOP Programming) • Công nghệ phần mềm (Software Engineering) • CSDL (Database) • Công nghệ Web (Web Technology): HTML/CSS, Javascript, Request-Response Model, MVC Model • Backend : nắm vững một/vài nền tảng như Java/.Net/PHP/Ruby • Frontend : JQuery, nắm vững một/vài javascript frameworks phổ biến như: React, Redux, Angular. • Kiến trúc phần mềm: nắm vững các lại kiến trúc phần mềm là điểm tốt, giúp bạn đi lên thành Software Architect.
Kỹ năng	• Anh Văn (English) • Suy nghĩ luận lý (Logical Thinking) • Giải quyết vấn đề (Problem Solving) • Quản lý công việc và thời gian (Task & Time Management) • Làm việc nhóm (Teamwork) • Thuyết trình và giao tiếp (Presentation & Communication)

Ngoài ra, ĐAM MÊ, ham học hỏi, khổ luyện, kiên trì, và không ngừng phấn đấu là những yếu tố cực kỳ quan trọng cho sự thành công trong nghề SD.

Các nấc thang mà bạn phải leo để lên đến đỉnh cao của nghề SD (thời gian leo ở mỗi nấc tùy theo nỗ lực và sự kiên trì của bạn).

1 **Fresher SD**	Bạn mới vào nghề SD, thường bạn sẽ tham gia vô 1 dự án đang chạy để hỗ trợ các senior SD. Nhiệm vụ của bạn lúc này là: • Học hỏi để nắm rõ hết nghiệp vụ của tất cả dự án bạn tham gia • Học các công nghệ được dùng trong tất cả dự án bạn tham gia • Học luôn kiến thức/kinh nghiệm của các senior SD

2 **SD**	Ở mức này bạn đã tích lũy được một số kiến thức về nghiệp vụ và công nghệ, và có kinh nghiệm lập trình tương đối. Nhiệm vụ của bạn lúc này là: • Học hỏi để nắm rõ hết nghiệp vụ của tất cả dự án bạn tham gia • Học các công nghệ được dùng trong tất cả dự án bạn tham gia • Học luôn kiến thức/kinh nghiệm của các senior SD
3 **Senior SD**	Ở mức này bạn đã có kiến thức về nghiệp vụ khá. Trình độ kỹ thuật vững trên một số công nghệ cụ thể để có thể tham gia 1 dự án từ đầu. Nhiệm vụ của bạn là: • Vẫn tích lũy thêm kiến thức nghiệp vụ • Biết cách phân chia việc cho các SD khác phụ mình (nếu cần) • Học hỏi để nắm vững các kiến trúc phần mềm của các dự án bạn đã làm để có thể xây dựng 1 hệ thống IT từ A-Z
4 **Software Architect (SA)**	Lên mức này thì bạn đã có thể xây dựng kiến trúc 1 hệ thống IT từ đầu. Nhiệm vụ của bạn là: • Quan tâm đến các **yêu cầu phi chức năng (Non-Functional Requirement – NFR)**, phổ biến là Performance, Availability, Scalability, and Security khi triển khai 1 hệ thống IT đến người dùng. • Vẫn tích lũy thêm kiến thức nghiệp vụ • Cập nhật sự phát triển công nghệ
5 **CTO (Chief Technology Officer)**	Ở mức này bạn chịu trách nhiệm mọi vấn đề liên quan đến công nghệ của công ty và nhân sự IT. Bạn vẫn luôn: • Tích lũy thêm kiến thức nghiệp vụ • Cập nhật sự phát triển công nghệ

4) Kiểm tra chất lượng sản phẩm (Quality Control – QC)

Nghề QC quản lý chất lượng sản phẩm phần mềm trước khi đưa đến tay người dùng.
Trong khi BA làm nhiệm vụ **validate phần mềm**, thì QC làm nhiệm vụ **verify phần mềm**, nói 1 cách nôm na là:

- **Validate (Ensure do the right thing)**: đảm bảo làm đúng cái yêu cầu (ví dụ làm con rô-bốt chó thì phải ra là rô-bốt chó chứ không được ra rô-bốt mèo).
- **Verify (Ensure do the thing right)**: đảm bảo cái yêu cầu cần làm được thực hiện chính xác, không có lỗi (ví dụ làm rô-bốt chó thì phải sủa gâu gâu chứ không thể sủa meo meo)

Nói một cách khác, QC sẽ làm cách gì đó để tìm ra các **lỗi (bug or defect)**, sau đó thông báo với bên SD để họ **sửa lỗi (fix bug)**, hoặc QC cũng có thể đưa ra các **phản hồi, gợi ý (feedbacks, comments)** để làm cho phần mềm tốt hơn cho người dùng.

Những công việc mà QC thực hiện trong SCRUM team:

- QC sẽ lên các **kế hoạch kiểm tra phần mềm (Test Plan)** ứng với các mục đích cần kiểm tra trong từng giai đoạn phát triển phần mềm. Mỗi Test Plan có các **trường hợp kiểm tra (Test Case)**.
- Sau khi có Test Plans và Test Cases, QC có thể tiến hành kiểm tra phần mềm bằng 2 cách:
  - **Kiểm tra bằng tay (Manual Test)**: QC sẽ thực hiện các Test Cases bằng cách tương tác trực tiếp với phần mềm.
  - **Kiểm tra tự động (Automation Test):** QC sẽ viết các **kịch bản (scripts)** để thực hiện các Test Cases một cách tự động: có nghĩa là các kịch bản này sẽ tự động tương tác với phần mềm, nhập dữ liệu, bấm nút này nút kia… mà QC không phải đụng tay đến.

- Một cách trực quan ta cũng thấy Automation Test hiệu quả hơn Manual Test ở thời gian chạy Test Cases và xuất kết quả, tuy nhiên để thực hiện Automation Test thì QC phải biết viết scripts sử dụng ngôn ngữ lập trình hoặc các công cụ hỗ trợ khác.

Trong SCRUM, chu trình phát triển phần mềm được chia nhỏ thành những đoạn nước rút (gọi là Sprint).

- Khi 1 Sprint bắt đầu, trong khi SD viết code để thực hiện các chức năng của phần mềm thì QC cũng bắt đầu viết Test Plans & Test Cases.
- Khi SD xong các tính năng thì QC bắt đầu chạy Test Cases để kiểm tra, nếu phát hiện lỗi thì sẽ báo cho SD để sửa.
- SD sửa xong các lỗi do QC báo thì QC lại chạy lại Test Cases để đảm bảo các lỗi đã được sửa.
- Nếu tất cả Test Cases đều ok thì Sprint hoàn thành và các tính năng phần mềm đã được QC kiểm tra sẽ được đưa đến người dùng.

Để làm QC tốt thì cần có những đặc điểm sau:

Tính tình	• Thích tìm lỗi của sản phẩm và tìm cách cải tiến sản phẩm • Cẩn thận & Tỉ mỉ • Trách nhiệm
Kiến thức	• Công nghệ kiểm tra phần mềm (Software Testing) • Công nghệ phần mềm (Software Engineering) : điểm + • Kiến thức nghiệp vụ chuyên ngành càng nhiều càng tốt như: o **Hệ quản trị doanh nghiệp (Enterprise Resource Planning – ERP)** o **Thương mại điện tử (e-Commerce)** o **Tiếp thị số (Digital Marketing)** o **Ngân hàng (Banking)…** Những cái này sẽ được tích lũy dần qua từng dự án.
Kỹ năng	• Anh Văn (English) • Suy nghĩ luận lý (Logical Thinking) • Phân tích vấn đề (Problem Analysis) • Phân vai: nghĩa là QC sẽ đóng vai trò là người dùng phần mềm để chạy xem phần mềm thế nào dưới góc nhìn của người dùng

Nếu bạn xuất phát điểm là SD mà thích thì bạn cũng có thể chuyển qua làm QC.

Bạn có thể làm Automation Test ngon lành cành đào.

Các nấc thang mà bạn phải leo để lên đến đỉnh cao của nghề QC (thời gian leo ở mỗi nấc tùy theo nỗ lực và sự kiên trì của bạn).

1 Fresher QC	Bạn mới vào nghề QC, thường bạn sẽ tham gia vô 1 dự án đang chạy để hỗ trợ các senior QC. Nhiệm vụ của bạn lúc này là: • Học hỏi để nắm rõ hết nghiệp vụ của tất cả dự án bạn tham gia • Học hỏi kiến thức/kinh nghiệm của các senior QC.
2 QC	Ở mức này bạn đã có kiến thức về nghiệp vụ của khá nhiều lĩnh vực và có kinh nghiệm QC tương đối vững để có thể tham gia 1 dự án từ đầu. Nhiệm vụ của bạn là: • Tích lũy thêm kiến thức lĩnh vực • Tích lũy kinh nghiệm QC • Học hỏi kiến thức/kinh nghiệm của các senior QC. Ghi chú: Nếu bạn có khả năng lập trình thì có thể cân nhắc bắt đầu tìm hiểu về **Automation Test** để chuyển sang làm Auto QC.
3 Senior QC	Ở mức này bạn đã có kiến thức về nghiệp vụ và có kinh nghiệm QC đầy mình. Nhiệm vụ của bạn là: • Vẫn tích lũy thêm kiến thức lĩnh vực • Biết cách phân chia việc cho các QC khác phụ mình (nếu cần) • Cập nhật công nghệ nếu bạn làm Auto QC

5) Kiểm tra chất lượng qui trình (Quality Assurance – QA)

QA (Quality Assurance) và QC (Quality Control) khi dịch qua tiếng Việt đều có nghĩa là kiểm tra chất lượng, tuy nhiên vai trò của 2 nghề này khác nhau:

- QC kiểm tra chât lượng của phần mềm.
- QA kiểm tra chất lượng của **qui trình phát triển phần mềm (Software Development Process)**.

QA là vị trí thường có trong các ITOC, nơi mà QA sẽ kiểm tra:

- Công ty có theo đúng các chứng chỉ về qui trình phát triển phần mềm như **ISO, CMMI** mà họ đã đạt được (nếu có) hay không
- Các nhóm làm phần mềm có theo các chuẩn phát triển phần mềm như **tiêu chuẩn về viết code (Coding Standard)**, **qui định viết code (Coding Convention)**, **định dạng code (Coding Style)**… hoặc qui trình về **xem lại code (code review)**, nôm na là sau khi một SD phát triển một tính năng gì đó thì phần code bạn này đã làm sẽ được review bởi một SD khác hoặc Team Lead để đảm bảo bạn này làm đúng tính năng phần mềm theo yêu cầu và đúng qui trình trước khi phần code này được bỏ vào hệ thống.

Coding standard có thể có 2 loại đồng thời:

- Một loại khách hàng cung cấp bắt công ty phải theo để họ an tâm hơn về chất lương.
- Một loại công ty đưa ra cho các đội làm theo nhằm đảm bảo tính thống nhất cho toàn công ty.

Dự án nào mà khách hàng đưa coding standard thì đội làm dự án đó phải kết hợp 2 bộ coding standard để đưa ra 1 bộ tổng hợp cho các thành viên trong đội làm theo. QA sẽ kiểm tra **(audit)** trên bộ tổng hợp này.

Các ITOC hiện nay hầu hết theo **qui trình phát triển phần mềm nhanh (Agile Methodology)** mà một mô hình phổ biến nhất hiện nay là **SCRUM.** The mô hình này thì qui trình phát triển một phần mềm sẽ được chia nhỏ ra thành nhiều **bước chạy nhanh (Sprint)**, QA sẽ audit các đội trên từng Sprint.

- Trước mỗi Sprint thì QA sẽ liệt kê ra một số danh sách các **tiêu chí (criteria)** mà họ sẽ audit gửi đến đội dự án.
- Sau khi Sprint kết thúc thì QA sẽ ngồi lại với PM của đội dự án để duyệt qua từng criterion xem đội hoàn thành như thế nào?
  - Nếu đội hoàn thành tốt thì kết quả là **C (Compliant)**
  - Nếu đội vi phạm thì kết quả là **NC (Non-Compliant)**. Đội nào làm mà không/ít NC thì cuối năm sẽ có thể có thêm bonus, ngược lại thì sẽ không có chút cháo gì ăn Tết.

Để làm QA tốt thì cần có những đặc điểm sau:

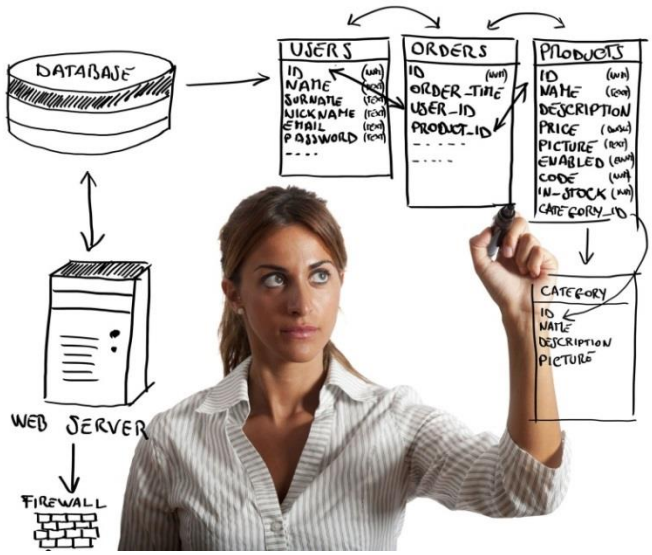
Tính tình	• Nghiêm chỉnh, luôn theo qui trình • Tỉ mỉ
Kiến thức	• Các qui trình phát triển phần mềm: Waterfall, Agile • Các chuẩn/chứng chỉ về qui trình phát triển phần mềm: ISO, CMMI • Công nghệ phần mềm (Software Engineering): (điểm +)
Kỹ năng	• Anh Văn (English) • Suy nghĩ luận lý (Logical Thinking) • Đặc tả các tiêu chí để audit • Audit • Giao tiếp (Communication)

Nghề QA không yêu cầu kiến thức chuyên môn về IT nhiều nên ai thích thì đều có thể theo đuổi.

Các nấc thang mà bạn phải leo để lên đến đỉnh cao của nghề QA (thời gian leo ở mỗi nấc tùy theo nỗ lực và sự kiên trì của bạn).

1 Fresher QA	Bạn mới vào nghề QA, thường bạn sẽ tham gia vô 1 dự án đang chạy để hỗ trợ các senior QA. Nhiệm vụ của bạn lúc này là: • Học hỏi để nắm rõ hết qui trình/ chứng chỉ phát triển phần mềm của tất cả dự án bạn tham gia • Học hỏi kiến thức/kinh nghiệm của các senior QA.
2 QA	Ở mức này bạn đã có kiến thức về các qui trình/chứng chỉ tương đối vững để có thể tham gia 1 dự án từ đầu. Nhiệm vụ của bạn là: • Tích lũy thêm kiến thức qui trình/chứng chỉ • Tích lũy kinh nghiệm QA • Học hỏi kiến thức/kinh nghiệm của các senior QA.
3 Senior QA	Ở mức này bạn đã có kiến thức về qui trình/chứng chỉ và có kinh nghiệm QA đầy mình. Nhiệm vụ của bạn là: • Vẫn tích lũy thêm kiến thức qui trình/chứng chỉ • Biết cách phân chia việc cho các QA khác phụ mình (nếu cần)

6) Quản trị CSDL (Database Admin – DBA)



DBA, như cái tên đã gợi ý, có nhiệm vụ quản trị CSDL trong hệ thống IT của doanh nghiệp. Hệ thống này có thể bao gồm nhiều phần mềm kết nối với nhau hoặc kết nối với nhiều CSDL khác nhau. Do đó, trách nhiệm của DBA là đảm bảo các CSDL này hoạt động tốt, đáp ứng yêu cầu dữ liệu từ các phần mềm.

Trước đây, CSDL là **CSDL quan hệ** (**relational database**) trong đó các bảng dữ liệu quan hệ với nhau qua khóa ngoại (**foreign key**), số lượng quan hệ càng lớn thì CSDL càng phức tạp, DBA phải đảm bảo dữ liệu phải được toàn vẹn (**data integrity**).
Khi lượng dữ liệu lưu trữ trở nên to lớn thì tốc độ truy xuất sẽ bị ảnh hưởng đáng kể. DBA sẽ phải dùng một số kỹ thuật, như đánh chỉ số (**indexing**), để giúp việc truy vấn dữ liệu nhanh hơn trên các bảng dữ liệu lớn.

Trong các hệ thống phần mềm mà CSDL không quá phức tạp hoặc không quá lớn, vị trí DBA sẽ được Software Developer (SD) kiêm luôn. Nhưng khi CSDL ngày càng phức tạp và số lượng dữ liệu ngày càng lớn ảnh hưởng đến **năng lực của hệ thống** (**system performance**) thì phải cần DBA.

Một hệ thống IT bị chậm có thể do nhiều nguyên nhân gây nên. Trong số các lý do thì việc truy vấn dữ liệu từ máy chủ (**web server/application server**) đến hệ quản trị CSDL (**DBMS – Database Management System**) bị chậm cũng khá phổ biến.

Việc truy vấn chậm này có thể do 2 yếu tố:

- Câu truy vấn do SD viết bằng **SQL** không tốt (hoặc phần lập trình nối CSDL không tốt)
- HOẶC/VÀ CSDL được DBA quản trị không tốt.

Nếu SD không có kiến thức về CSDL và DBA không có kiến thức về lập trình thì có khả năng SD & DBA sẽ đổ lỗi cho nhau; tệ hơn nữa là SD cứ cố gắng tối ưu hóa (**optimize**) câu truy vấn/phần lập trình kết nối CSDL trong khi lỗi do quản trị CSDL không tốt và ngược lại là DBA cứ cố gắng tối ưu hóa CSDL trong khi phần truy vấn viết vô cùng củ chuối !!!

Một số công việc chính mà DBA cần thực hiện hàng ngày/tuần/tháng/quý/năm như:

- Đánh giá khả năng phần cứng của máy chủ CSDL
- Cài phần mềm CSDL trên các nền HĐH khác nhau
- Xây dựng, thực thi chính sách bảo mật thông tin
- Tạo mới và nâng cấp bản mới cho CSDL
- Thực hiện giải pháp sao lưu dữ liệu, phòng ngừa thảm họa
- Phục hồi dữ liệu khi bị hư hoặc bị mất
- Quản trị người dùng truy nhập CSDL
- Theo dõi và tối ưu tinh chỉnh CSDL
- Thường xuyên cập nhật kiến thức, giải pháp công nghệ mới

Để làm DBA tốt thì cần có những đặc điểm sau:

Tính tình	• Thích làm việc với dữ liệu • Cẩn thận & Tỉ mỉ • Trách nhiệm
Kiến thức	• Thiết kế cơ sỡ dữ liệu • Kiến thức về các hệ quản trị CSDL phổ biến (Oracle, MS SQL, MySQL…) • Công nghệ phần mềm (điểm +) • Lập trình (điểm +)
Kỹ năng	• Anh Văn (English) • Quản lý dữ liệu (Data Management) • Phân tích tổng hợp (Synthesis Analysis) • Chẩn đoán vấn đề (Problem Diagnosis) • Làm việc nhóm (Teamwork)

Các nấc thang mà bạn phải leo để lên đến đỉnh cao của nghề DBA (thời gian leo ở mỗi nấc tùy theo nỗ lực và sự kiên trì của bạn).

1 Fresher DBA	Bạn mới vào nghề DBA, thường bạn sẽ tham gia vô 1 dự án đang chạy để hỗ trợ các senior DBA. Nhiệm vụ của bạn lúc này là: • Học hỏi để nắm rõ hết các **lược đồ quan hệ thực thể (Entity Relationship Diagram – ERD)** của tất cả dự án bạn tham gia • Học hỏi để nắm rõ hết các **hệ quản trị CSDL (Database Management System – DBMS)** của tất cả dự án bạn tham gia • Học hỏi kiến thức/kinh nghiệm của các senior DBA.
2 DBA	Ở mức này bạn đã có kiến thức về các ERD/ DBMS tương đối vững để có thể tham gia 1 dự án từ đầu. Nhiệm vụ của bạn là: • Tích lũy thêm kiến thức ERD • Tích lũy thêm kiến thức DBMS • Tích lũy kinh nghiệm DBA • Học hỏi kiến thức/kinh nghiệm của các senior DBA.
3 Senior DBA	Ở mức này bạn đã có kiến thức về ERD/DBMS và có kinh nghiệm DBA đầy mình. Nhiệm vụ của bạn là: • Vẫn tích lũy thêm kiến thức ERD • Vẫn tích lũy thêm kiến thức DBMS • Biết cách phân chia việc cho các DBA khác phụ mình (nếu cần)

7) Khoa học dữ liệu (Data Scientist)

Data Scientist, tạm dịch là Nhà khoa học dữ liệu, nôm na là người làm trong Ngành khoa học dữ liệu (**Data Science**) với các công việc trên dữ liệu: thống kê, phân tích, xử lý… để tìm ra các thông tin hữu ích từ dữ liệu nhằm đáp ứng các nhu cầu của danh nghiệp.

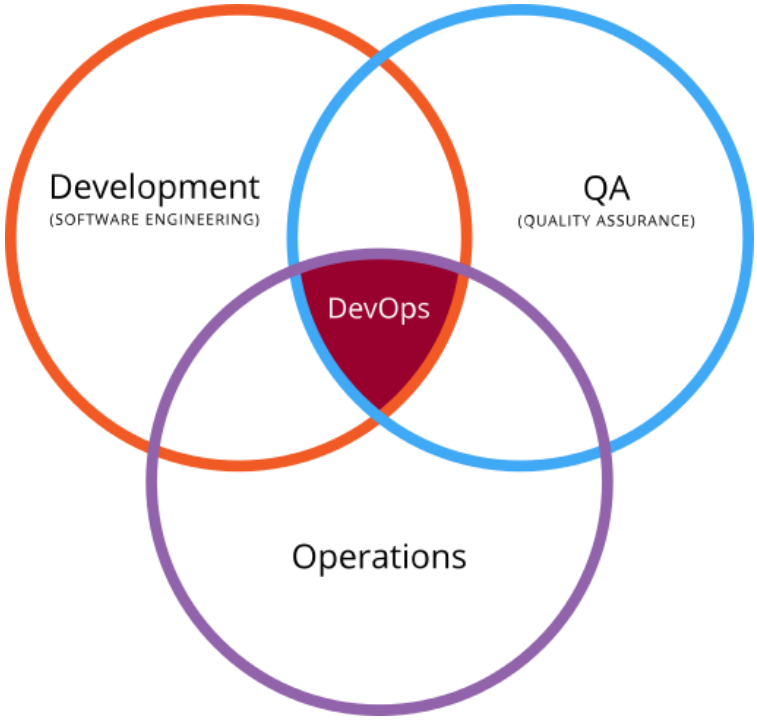
Nghề khoa học dữ liệu có thể được chia ra làm 3 nhành nhỏ: Xử lý dữ liệu (**Data Analytics**), Khai thác dữ liệu (**Data Mining**) và Kho dữ liệu (**Data Warehouse**) đã được đặc tả ở trên.

Để làm Data Scientist tốt thì cần có những đặc điểm sau:

Tính tình	• Thích phân tích dữ liệu • Cẩn thận • Tỉ mỉ
Kiến thức	• Toán (Maths) • Thống kê (Statistics) • Lập trình (Programming): điểm + • Cở sỡ dữ liệu (Database) • Kiến thức các chuyên ngành (Domain Knowledge)
Kỹ năng	• Anh Văn (English) • Suy nghĩ luận lý (Logical Thinking) • Mô hình hóa dữ liệu (Data Visualization) • Phân tích dữ liệu (Data Analysis) • Xử lý dữ liệu (Data Processing)

Nghề Data Scientist nói chung là khá hấp dẫn, tuy nhiên khá chuyên biệt nên số lượng công việc không nhiều so với các nghề IT khác.

8) Triển khai vận hành hệ thống (DevOps)



Lúc trước chúng ta nghe nhiều về nghề **Quản trị mạng (Network Administrator)** hoặc **vận hành hệ thống (System Operator)** trong các công ty có hệ thống IT. Gần đây 2 nghề này ít nghe hơn, thay và đó thì nghề DevOps bắt đầu phổ biến.

Như hình ở trên đã mô tả thì nghề DevOps ngoài việc quản trị và vận hành hệ thống như trước kia thì còn tham gia vào quá trình phát triển phần mềm cũng như qui trình đảm bảo chất lượng phần mềm.

Cùng với sự ra đời của từ "DevOps" thì 2 từ tương ứng cũng ra đời gắn với nhiệm vụ của DevOps, đó là:

- **CI (Continuous Integration)**: liên tục tích hợp những phần nhóm phát triển đã xây dựng
- **CD (Continuous Delivery)**: liên tục triển khai **phiên bản (version)** mới cho người dùng

Khác với Network Admin, DevOps sẽ phối hợp với các vai trò khác (BA, SD, QC…) để xử lý sự cố xảy ra trong hệ thống IT. Sự cố có thể là do phần mềm, phần cứng, network, bảo mật…

Để làm DevOps tốt thì cần có những đặc điểm sau:

Tính tình	• Thích quản trị hệ thống IT • Kỉ luật & Tỉ mỉ • Trách nhiệm
Kiến thức	• Mạng máy tính (Computer Network): Internet, Intranet, WAN, LAN… • HĐH (Operating System): UNIX, Linux, Windows … • Công nghệ phần mềm (Software Engineering): điểm + • Cở sỡ dữ liệu (Database): MySQL, MSSQL, NoSQL… • Các công cụ thao tác hệ thống: telnet, putty, ssh… • Lập trình (Programming): điểm +
Kỹ năng	• Anh Văn (English) • Suy nghĩ luận lý (Logical Thinking) • Giải quyết vấn đề (Problem Solving) • Quản trị mạng (Network Administration) • Phân tích sự cố (Troubleshooting) • Làm việc nhóm với SDs khi giải quyết sự cố (Team Work)

Các nấc thang mà bạn phải leo để lên đến đỉnh cao của nghề DevOps (thời gian leo ở mỗi nấc tùy theo nỗ lực và sự kiên trì của bạn).

1 Fresher	Bạn mới vào nghề DevOps, thường bạn sẽ tham gia vô 1 dự án đang chạy để hỗ trợ các senior DevOps.Nhiệm vụ của bạn lúc này là: • Học hỏi để nắm rõ hết các **lược đồ vật lý (Physical Diagram)** - thể hiện cách triển khai hệ thống IT -, CI/CD của tất cả dự án bạn tham gia • Học hỏi kiến thức/kinh nghiệm của các senior DevOps.
2 DevOps	Ở mức này bạn đã có kiến thức về các Physical Diagram và kinh nghiệm DevOps tương đối nhiều để có thể tham gia 1 dự án từ đầu. Nhiệm vụ của bạn là: • Tích lũy thêm kiến thức OS, Network, Cloud, Physical Diagram, CI/CD • Tích lũy kinh nghiệm DevOps • Học hỏi kiến thức/kinh nghiệm của các senior DevOps.
3 Senior DevOps	Ở mức này bạn đã có kiến thức và kinh nghiệm đầy mình. Nhiệm vụ của bạn là: • Vẫn tích lũy thêm kiến thức OS, Network, Cloud, Physical Diagram • Biết cách phân chia việc cho các DevOps khác phụ mình (nếu cần)

9) Quản lý dự án (Project Manager – PM)

Trong bất cứ một dự án nào cũng phải có một người hay một nhóm người quản lý dự án. Vì vậy nghề Project Manager (PM) hiểu nôm na là người giúp dự án vận hành trơ tru từ khâu chuẩn bị yêu cầu sản phẩm, phân tích, thiết kế, đến hiện thực, kiểm tra, triển khai, hỗ trợ khách hàng và bảo trì.

Trong dự án phần mềm, PM là người điều phối qui trình phát triển phần mềm liên quan đến tất cả các nghề IT mà tôi đã chia sẻ ở trên: IT Support, Business Analyst, Software Developer, Quality Control, Quality Assurance, Database Admin, Data Scientist, and DevOps.

Đây là vị trí về quản lý cho nên để đảm nhận được vị trí này thì bạn phải có kiến thức và kinh nghiệm nhiều đáng kể trong lĩnh vực phát triển phần mềm. Tuy nhiên, kiến thức, kỹ năng và kinh nghiệm cho vị trí PM sẽ khác nhau cũng khá lớn ch 2 mô hình công ty:

- **SMEs & TSs**: PM ở đây thường là người rất mạnh về kỹ thuật, thường đi lên từ SD và học thêm về quản lý dự án/con người.

- **ITOC**: PM ở đây có thể không cần mạnh về kỹ thuật, phần kỹ thuật có vị trí TL (Team Lead) lo, hiển nhiên PM biết về kỹ thuật là một lợi thế lớn giúp họ quản lý dự án tốt hơn. PM ở đây thường đi lên từ QC/ SD và học thêm về quản lý dự án/con người hoặc một người có kiến thức/kỹ năng/kinh nghiệm về quản lý cũng có thể vô làm PM, họ cần học thêm tí kiến thức IT cũng đủ.

Để làm PM tốt thì cần có những đặc điểm sau:

Tính tình	• Thích quản lý dự án và con người • Kỉ luật • Trách nhiệm
Kiến thức	• Kiến thức tổng quan về IT • Công nghệ phần mềm (Software Engineering) • Lập trình (Programming): điểm +
Kỹ năng	• Anh Văn (English) • Suy nghĩ luận lý (Logical Thinking) • Giải quyết vấn đề (Problem Solving) • Giao tiếp (Communication) • Thương thảo (Negotiation) • Quản lý thời gian (Time Management) • Làm việc nhóm (Team Work)

Nghề PM chỉ có 2 nấc thang là PM và Senior PM.

III. Căn bản lập trình

1) Ngôn ngữ & mô hình lập trình (Programming Language & Paradigms)

Trong thế giới tự nhiên thì các đối tượng giao tiếp với nhau dựa trên ngôn ngữ: con người nói chuyện với nhau dùng tiếng người, con vật nói chuyện với nhau dùng tiếng của loài đó. Con người nói chuyện với con vật thì con người nói giả tiếng của nó.

Tương tự trong thế giới IT, con người nói chuyện với MVT thì cũng cần có ngôn ngữ gọi là **ngôn ngữ lập trình (NNLT)**. Con người "nói" cho MVT hiểu thông qua các **mã (code)** được định nghĩa bởi các NNLT.

Người viết các mã để ra lệnh cho MVT gọi là **Lập trình viên (Programmer) – LTV**. Một số nghề IT ở trên đòi hỏi phải biết lập trình như SD, QC (automation), DBA, DevOps.

Lịch sử phát triển của NNLT như sau:

- Ngôn ngữ máy (Machine Language): là những đoạn mã mà mỗi phần tử của nó có giá trị 0 hoặc 1, ví dụ 00101011, mà MVT hiểu được.
- Ngôn ngữ Assembly (Assembly Language): ngôn ngữ được phát triển dựa trên ngôn ngữ máy giúp con người dễ hiểu và dễ nhớ hơn khi lập trình.
- Ngôn ngữ Pascal (Pascal Language): ngôn ngữ được thiết kế ở mức cao hơn nữa giúp con người dễ hiểu và dễ nhớ hơn Assembly. Hồi trước ngôn ngữ này được dùng phổ biến để dạy lập trình chứ ít được dùng để xây dựng phần mềm thực tế.
- Ngôn ngữ cấp cao như C/C++, Java, C#, PHP: các **NNLT tổng quát (general programming language)** khá phổ biến này được sử dụng để xây dựng phần mềm thực tế.
- Ngoài ra có khá nhiều NNLT khác như Ruby, Python, Perl, Go… được tạo ra để phục vụ cho việc phát triển các phần mềm trong một lĩnh vực cụ thể nào đó nhằm giảm thời gian và chi phí phát triển đáng kể.

Các NNLT chỉ khác nhau về **cú pháp (syntax)**, tức là cách viết theo thứ tự nào đó để MVT hiểu, nên việc SD chuyển từ NNLT này sang NNLT khác trên lý thuyết không vấn đề. Nhưng thực tế thì SD sẽ mất thời gian đáng kể để chuyển đổi NNLT bởi vì không hẳn bản thân NNLT mà là các công ty đứng sau lưng phát triển các **thư viện (Library)** và **khung làm việc (Framework – FW)** cho NNLT đó.

Tuy nhiên, các thư viện/FWs đều có những điểm chung dựa trên các **khái niệm (Concept)** trong lập trình, vì vậy nếu SD nắm vững các khái niệm này thì sẽ rút ngắn được khá nhiều thời gian chuyển đổi NNLT.

Khi một NNLT được định nghĩa ra thì đi kèm với nó sẽ là một chương trình dịch có chức năng dịch NNLT này sang ngôn ngữ máy để MVT hiểu và thực thi.
Có 2 loại chương trình dịch:

- **Trình biên dịch (Compiler)**: Đây là chương trình sẽ dịch toàn bộ các files của **lập trình viên (Programmer) – LTV** từ đầu đến cuối rồi mới thực thi. Ví dụ: ngôn ngữ Java, C#.
- **Trình thông dịch (Intepreter)**: Chương trình này thì vừa dịch vừa thực thi các files của LTV từng dòng một. Ví dụ: ngôn ngữ Javascript.

Loại chương trình dịch cũng có ảnh hưởng đến việc lập trình đáng kể. Một ví dụ cơ bản điển hình dễ thấy nhất là khai báo biến số.

- Đối với NNLT dùng trình biên dịch: Ta có thể khai báo biến số x ở cuối chương trình và xài x ở đầu chương trình.
- Đối với NNLT dùng trình thông dịch: Ta phải khai báo biến số x trước khi xài.

Qui trình xây dựng 1 chương trình dịch gồm các bước sau:

- **Phân tích từ vựng (Lexical Analysis)**: bước này phân tích các **phần tử (token)** được định nghĩa trong NNLT.
  Ví dụ: **var a = b + 1;** → tokens = keyword (‘var’), id (‘a’, ‘b’), op (‘=’, ‘+’), num (‘1’).
- **Phân tích cú pháp (Syntactical Analysis)**: bước này phân tích thứ tự các tokens xuất hiện trong 1 lệnh (statement) của NNLT có đúng như định nghĩa của NNLT đó hay không? Ví dụ trên là đúng cú pháp của ngôn ngữ Javascript, sai cú pháp của ngôn ngữ Java vì ngôn ngữ này không định nghĩa cú pháp như ví dụ, cú pháp đúng cho phép gán được định nghĩa của Java sẽ là: **int a = b + 1;.** Như vậy khi lập trình thì LTV phải biết mình đang sử dụng NNLT gì để viết cho đúng cú pháp của NNLT đó.
- **Xử lý ngữ nghĩa (Semantic Processing)**: bước này xử lý các lệnh của NNLT có ý nghĩa gì. Như ví dụ trên thì lệnh đó có nghĩa là gán kết quả của phép cộng giữa biến số b và hằng số 1 vào biến số a.

Nếu bạn có khả năng xây dựng được 1 chương trình dịch theo qui trình trên thì bạn hoàn toàn có thể cho ra đời 1 NNLT mới, đặt tên của bạn chẳng hạn. Vấn đề là NNLT bạn tạo ra có đem lại lợi ích nhiều cho LTV để họ sử dụng NNLT của bạn hay không.

Tùy NNLT định nghĩa mà LTV có thể sử dụng các loại kiểu dữ liệu sau:

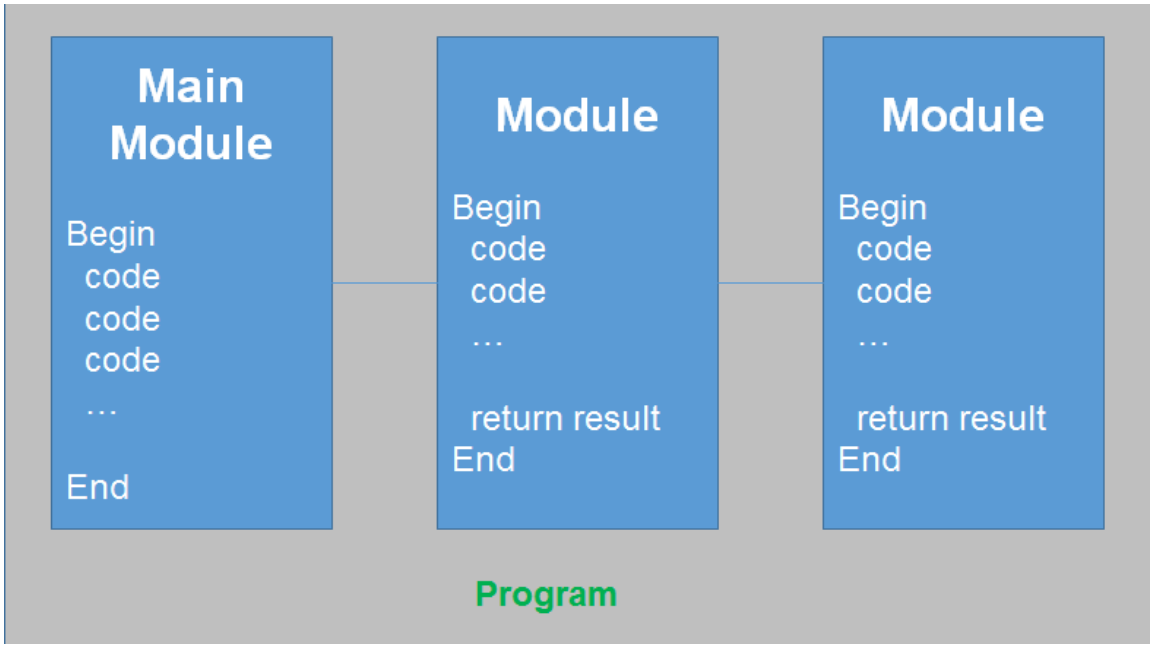
- **Kiểu dữ liệu cơ bản (Primitive Type)**: là các kiểu thông dụng như số nguyên, chuỗi…
- **Kiểu dữ liệu cấu trúc (Structured Type):** là kiểu dữ liệu có cấu trúc, mỗi phần tử của kiểu này có một kiểu dữ liệu riêng, có thể là kiểu dữ liệu cấu trúc nữa.
- **Kiểu dữ liệu con trỏ (Pointer Type)**: đây là kiểu dữ liệu khá đặc biệt, biến số thuộc kiểu này không chứa giá trị của biến số mà là chứa địa chỉ tham khảo của ô nhớ chứa giá trị. Vì vậy đây là kiểu dữ liệu rất linh hoạt nếu bạn biết cách xài, còn không thì bạn sẽ gặp rất nhiều rắc rối.

Các NNLT được thiết kế dựa trên các mô hình lập trình (Programming Paradigm) – MHLT, một số MHLT phổ biến sau:

- MHLT thủ tục
- MHLT hướng đối tượng
- MHLT khía cạnh
- MHLT hàm

Các MHLT này không loại trừ lẫn nhau, chúng có thể tồn tại đồng thời trong 1 NNLT.

Mô hình lập trình đầu tiên là **lập trình thủ tục (Procedure Programming)**:

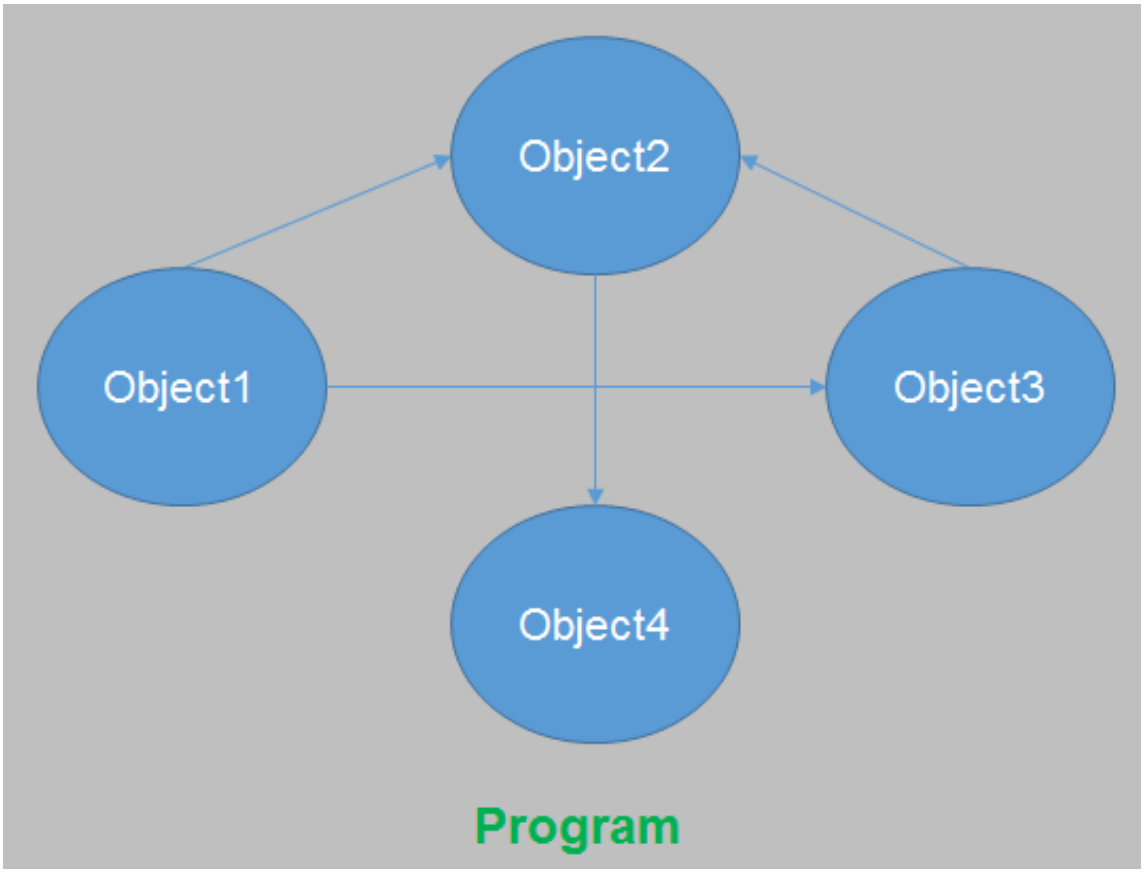


Trong lập trình thủ tục thì chúng ta có mô đun chính là **điểm vào (entry point)** của chương trình và trong quá trình thực thi thì các đoạn code trong mô đun chính có thể gọi các mô đun phụ khác để làm việc gì đó.

Nhược điểm của lập trình thủ tục là:

- Các mô đun không có khả năng che dấu các **thuộc tính (Property)** và **thủ tục (Procedure)** của nó nên bất kỳ mô đun nào cũng có thể truy cập được.
- Các mô đun không có sự kế thừa, khi ta viết một mô đun mới rất giống mô đun đã có trước đó (Properties, Procedures) thì ta phải viết lại hoặc copy rồi sửa các Properties, Procedures đó làm chúng ta tốn nhiều thời gian và sau này bảo trì khó khăn (sửa một mô đun nào đó thì cũng phải tìm và sửa hết các mô đun giống nó).
- Các mô đun không có kiểu (Type) và không có sự kế thừa nên không có được tính đa hình giúp chúng ta viết code rất linh hoạt.

Vì vậy, mô hình **lập trình hướng đối tượng (Object-Oriented Programming – OOP)** đã ra đời để giải quyết được những hạn chế của lập trình thủ tục.
Trong mô hình OOP, các mô đun được thay bằng các **đối tượng (Object)** thuộc các **lớp (Class)** hay ta gọi là các đối tượng có **kiểu (Type)**.



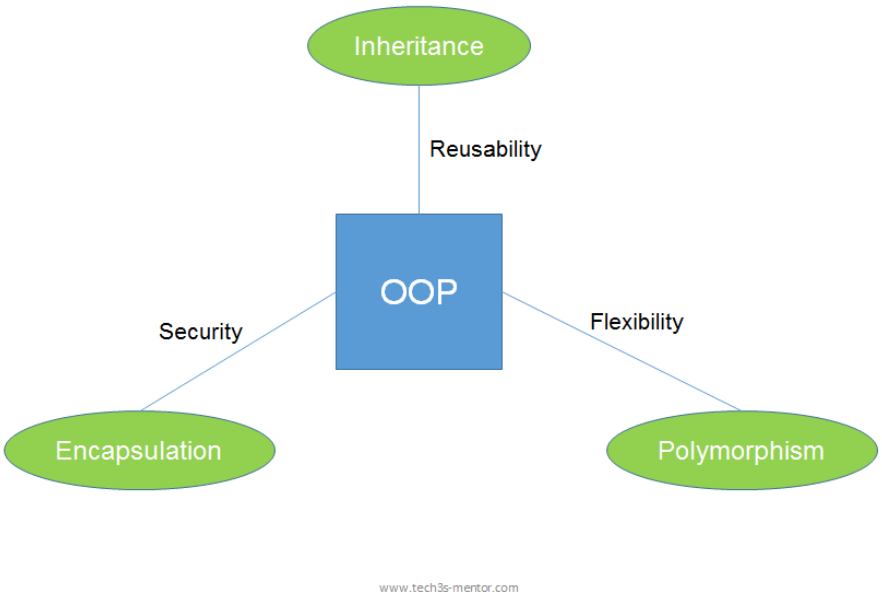
OOP có 3 thuộc tính cơ bản:

- **Bao đóng (Encapsulation)**: Đối tượng có khả năng che dấu các **thuộc tính (Property)** hoặc **phương thức (Method)** của nó để không cho các đối tượng khác truy cập được.
- **Thừa kế (Inheritance)**: Lớp con có thể thừa kế các thuộc tính và phương thức của lớp cha giúp giảm thời gian và công sức phát triển các lớp con.
- **Đa hình (Polymorphism)**: Cùng 1 phương thức được gọi bởi 1 kiểu có thể kích hoạt việc thực thi phương thức của các đối tượng khác nhau.

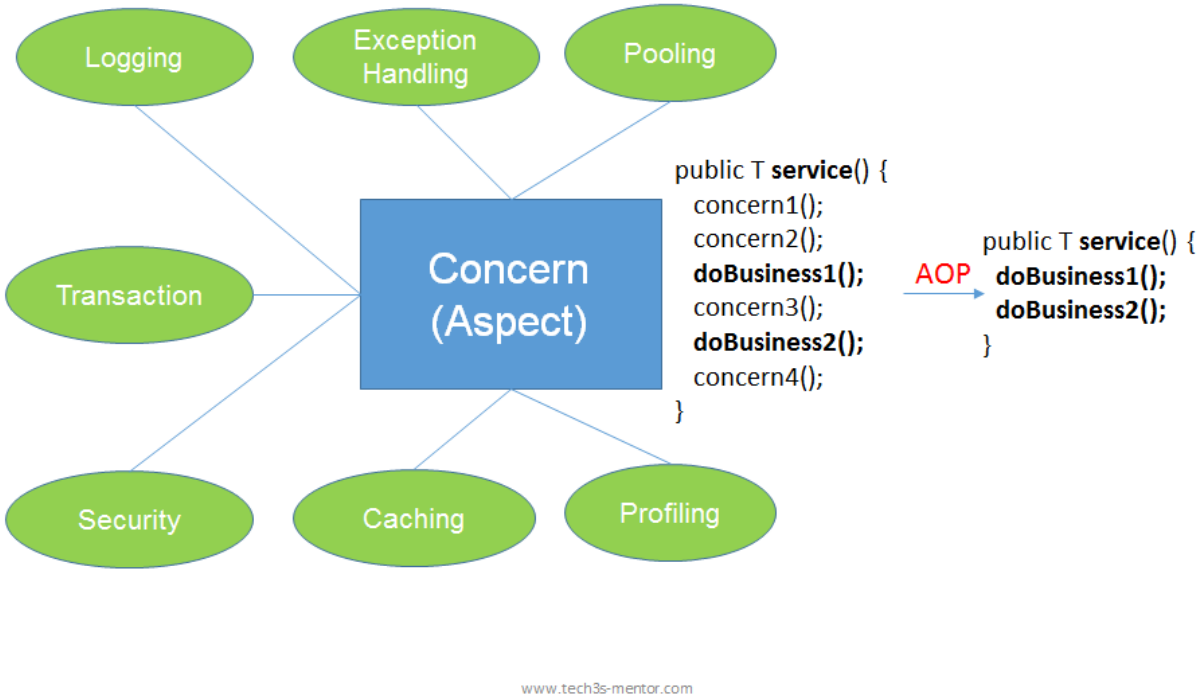
  Ví dụ: (Animal) animal.shout(); -> có thể kích hoạt phương thức shout() của Cat hoặc của Dog tùy lúc khởi tạo "animal" từ Cat hay Dog, trong đó Cat & Dog thừa kế Animal. Với tính đa hình này, ta không cần kiểm tra biến "animal" là Cat hay Dog rồi ép về kiểu Cat hay Dog rồi gọi phương thức shout().

Nhờ 3 thuộc tính cơ bản của OOP mà chương trình đạt được 3 tính chất quan trọng sau:

- **Tính bảo mật (Security)**
- **Tính tái sử dụng (Reusablity)**
- **Tính linh hoạt (Flexibility)**



Ngoài OOP, LTV cần nắm mô hình **lập trình khía cạnh (Aspect-Oriented Programming)**:



AOP giúp các đoạn code của ta gọn hơn, hiển thị đúng nghiệp vụ chính của phần mềm, giúp bạn dễ quản lý và bảo trì cũng như phát triển thêm các tính năng khác sau này hơn.

Ngoài OOP và AOP, bạn cũng cần nắm vững **lập trình hàm (Functional Programming)**, đại diện tiêu biểu là ngôn ngữ Javascript, trong Java có Lambda Expression, còn .Net thì có tính năng LINQ.

2) Cấu trúc dữ liệu và giải thuật (Data Structure & Algorithm)

Khi lập trình thì LTV có thể sử dụng các giải thuật trên các **cấu trúc dữ liệu (CTDL)** để giải quyết các vấn đề nào đó.

Ví dụ: LTV dùng giải thuật sắp xếp trên danh sách các hóa đơn để sắp xếp các hóa đơn theo thứ tự giá tiền từ nhỏ đến lớn. Để giải quyết vấn đề này thì LTV có thể sử dụng các giải thuật phổ biến như Bubble Sort, Quick Sort… (hoặc tự viết giải thuật riêng theo nhu cầu của mình) và CTDL có thể sử dụng là List.

CTDL đóng vai trò rất quan trọng trong lập trình vì nó giúp LTV mô hình hóa dữ liệu trong thế giới thực thành dữ liệu có cấu trúc trong MVT để giải thuật xử lý các dữ liệu này tự động.

Các loại CTDL thông dụng mà tất cả các NNLT đều hỗ trợ:

- **Danh sách (List)**: các phần tử được lưu tuần tự.
- **Tập hợp (Set)**: các phần tử được lưu tuần tự và không cho tồn tại 2 phần tử trùng nhau.
- **Cây (Tree)**: các phần tử được lưu theo hình cái cây, bắt đầu bằng **nút gốc (Root)** và đi xuống các **nút trung gian (Node)** đến tận cùng là **nút lá (Leaf)**.
- **Ánh xạ (Map)**: các phần tử được lưu tuần tự như danh sách, trong đó mỗi phần tử sẽ có 2 thành phần: **khóa (key) & giá trị (value)**. Giá trị này có thể là một con số hay một đối tượng nào đó, và cũng có thể là 1 danh sách, 1 tập hợp, 1 cây, hoặc thậm chí 1 ánh xạ.
  Ví dụ ta có thể lưu 1 danh sách các mã số lớp, mỗi mã số lớp (key) sẽ lấy ra 1 danh sách sinh viên (value):
  .[BKIT98-01 | [Hoàng, Hùng, Minh, Nga]]
  .[BKIT98-05 | [Phương, Thảo, Tiến]]

Trên mỗi loại CTDL, NNLT sẽ cung cấp sẵn một số lượng các giải thuật cơ bản để LTV tiện xài mà không cần phải tự suy nghĩ viết ra. NNLT nào khá hơn sẽ cung cấp thêm các giải thuật nâng cao khác. NNLT nào càng mạnh thì số lượng giải thuật càng nhiều cho LTV. Nếu LTV đang xài NNLT nào chưa hỗ trợ giải thuật nào đó họ cần thì họ có thể đi tìm các thư viện/FWs mà những LTV khác đã viết trước đó (chia sẻ cho cộng đồng hoặc bán).

Để lập trình nhanh và hiệu quả nhất thì LTV phải làm các việc sau:

- Nắm vững lý thuyết các CTDL cơ bản và một số CTDL nâng cao.
- Nắm vững các CTDL/giải thuật mà NNLT SD đang dùng hiện thực/cung cấp.
- Áp dụng các CTDL & giải thuật của NNLT này vào dự án thực tế càng nhiều càng tốt.

Khi chuyển sang NNLT mới thì LTV cần làm lại 2 mục sau cùng.

3) Tìm lỗi (Debugging)

Sau khi lập trình xong, bạn hồ hởi chạy thì kết quả không đúng như bạn mong đợi. Bạn phải làm sao?

Chuyện này là chuyện thường ngày ở huyện mà thôi. Bạn phải tìm cách gì đó để đảm bảo những phần code bạn đã viết chạy đúng ý bạn, cái này gọi là "**debug**". Thường thì có 2 cách để debug:

- Chạy từng dòng code trong **môi trường phát triển tích hợp (Integrated Development Environment – IDE).**
- In những thứ bạn muốn xem ra màn hình hoặc log files.

Cách số 1 chúng ta có thể làm trong lúc lập trình, nhưng khi chương trình đã đưa đến người dùng thì chúng ta chỉ có thể xài cách số 2.

Một cách trực quan thì cách số 1 dễ tìm ra lỗi hơn, dễ sửa lỗi rồi kiểm tra lại hơn và chi phí để sửa lỗi thấp hơn so với cách 2.

Kỹ năng tìm lỗi này rất quan trọng vì nó giúp chúng ta phát triển cũng như bảo trì phần mềm nhanh hơn và tốt hơn.

Phụ lục

* Danh sách các từ viết tắt

ALU	Athrimetical Logical Unit
AOP	Aspect-Oriented Programming
BA	Business Analyst
BE	Backend
DBA	Database Administrator
CNPM	Công Nghệ Phần Mềm
CPU	Central Processing Unit
CSDL	Cơ Sở Dữ Liệu
CTDL	Cấu Trúc Dữ Liệu
ERP	Enterprise Resource Planning
FE	Frontend
FW	Framework
HD	Hard Disk
HĐH	Hệ Điều Hành
HTML	Hyper Text Markup Language
ISP	Internet Service Provider
ITOC	IT Outsourcing Companie
LAN	Local Area Network
LTV	Lập Trình Viên
MHLT	Mô Hình Lập Trình
MMT	Mạng Máy Tính
MVT	Máy Vi Tính
NNLT	Ngôn Ngữ Lập Trình
OOP	Object-Oriented Programming
PM	Project Manager
PMNĐ	Phần Mềm Nguồn Đóng
PMNM	Phần Mềm Nguồn Mở
RAM	Random Access Memory
ROM	Read Only Memory
QA	Quality Assurance
QC	Quality Control
SA	Software Architect
SD	Software Developer
SME	Small and Medium Enterprise
SQL	Structured Query Language
TS	Tech Startup
UI	User Interface
URL	Uniform Resource Locator
US	User Story

* Các thuật ngữ Việt/Anh thông dụng cần nắm vững

Máy vi tính	**Computer**
Phần cứng	**Hardware**
Hệ điều hành	**Operating System**
Phần mềm	**Software**
Người dùng	**User**
Dữ liệu	**Data**
Cơ sở dữ liệu	**Database**
Mạng máy tính	**Computer Network**
Trí tuệ nhân tạo	**Artificial Intelligence**
Khoa học dữ liệu	**Data Science**
Bàn phím	**Keyboard**
Chuột	**Mouse**
Bộ xử lý trung tâm	**Central Processing Unit – CPU**
Màn hình	**Monitor**
Loa	**Speaker**
Máy in	**Printer**
Bộ nhớ chỉ đọc	**Read Only Memory – ROM**
Bộ nhớ truy cập ngẫu nhiên	**Random Access Memory – RAM**
Bộ nhớ ảo	**Virtual Memory**
Đĩa cứng	**Hard Drive**
Tập tin	**File**
Thư mục	**Directory**
Thu thập yêu cầu phần mềm	**Requirement Gathering**
Phân tích phần mềm	**Requirement Analysis**
Thiết kế phần mềm	**Software Design**
Hiện thực phần mềm	**Software Implementation**
Kiểm tra phần mềm	**Software Testing**
Triển khai phần mềm	**Software Deployment**
Bảo trì phần mềm	**Software Maintenance**
Phần mềm nguồn mở	**Open-source Software**
Phần mềm nguồn đóng	**Closed-source Software**
Tiến trình	**Process**

Tiểu tiến trình	**Thread**
Ngôn ngữ truy vấn có cấu trúc	**Structured Query Language – SQL**
Hệ quản trị CSDL	**DataBase Management System – DBMS**
CSDL quan hệ	**Relational Database**
Mô hình quan hệ thực thể	**Entity Relationship Diagram – ERD**
CSDL NoSQL	**NoSQL Database**
Giao thức	**Protocol**
Giao thức hỏi đáp	**Request/Response Protocol**
Máy chủ	**Server**
Máy khách	**Client**
Mạng cá nhân	**Personal Area Network – PAN (Bluetooth)**
Mạng nội bộ	**Local Area Network – LAN**
Mạng nội thành	**Metropolitant Area Network – MAN**
Mạng diện rộng	**Wide Area Network – WAN**
Mạng toàn cầu	**Internet**
Ngôn ngữ liên kết văn bản	**Hyper Text Markup Language – HTML**
Nhà cung cấp dịch vụ internet	**Internet Service Provider – ISP**
Địa chỉ IP	**IP Address**
Máy trung gian	**Gateway**
Dịch vụ định tên miền	**Domain Name Service**
Máy định tên miền	**Domain Name Server – DNS**
Công ty vừa và nhỏ	**Small and Medium Enterprises – SME**
Công ty gia công IT	**IT Outsourcing Companies – ITOC**
Công ty khởi nghiệp công nghệ	**Tech Startups - TS**
Hệ thống IT	**IT System**
Nhóm phát triển	**Development Team**
Vai trò	**Role**
Nhân viên	**Employee**
Nghề	**Job**
Chủ sản phẩm	**Product Owner – PO**
Nhà phân tích nghiệp vụ	**Business Analyst – BA**
Trưởng nhóm	**Team Lead – TL**
Nhà phát triển phần mềm	**Software Developer – SD**

Kiểm tra chất lượng sản phẩm	**Quality Control – QC**
Kiểm tra chất lượng qui trình	**Quality Assurance – QA**
Nhà quản trị CSDL	**Database Admin – DBA**
Nhà khoa học dữ liệu	**Data Scientist**
Chuyên viên triển khai vận hành hệ thống	**DevOps**
Nhà quản lý dự án	**Project Manager – PM**
Các luồng nghiệp vụ	**Business Flows**
Các qui luật nghiệp vụ	**Business Rules**
Các tính năng	**Features**
Các tương tác người dùng	**User Stories**
Xuất các tính năng sản phẩm	**Release**
Lỗi	**Bug or Defect**
Cải tiến hệ thống	**System Improvement**
Ứng dụng một trang	**Single Page App – SPA**
Lập trình ứng dụng di động	**Mobile Programming**
Yêu cầu phi chức năng	**Non-Functional Requirement – NFR**
Kế hoạch kiểm tra phần mềm	**Test Plan**
Trường hợp kiểm tra	**Test Case**
Kiểm tra bằng tay	**Manual Test**
Kiểm tra tự động	**Automation Test**
Qui trình phát triển phần mềm	**Software Development Process**
Tiêu chuẩn về viết code	**Coding Standard**
Qui định viết code	**Coding Convention**
Định dạng code	**Coding Style**
Xem lại code	**Code Review**
Nhà triển khai và vận hành hệ thống IT	**DevOps**
Liên tục tích hợp tính năng	**Continuous Integration – CI**
Liên tục triển khai phiên bản (version)	**Continuous Delivery – CD**
Ngôn ngữ lập trình	**Programming Language**
Lập trình viên	**Programmer**
Cú pháp	**Syntax**
Thư viện	**Library**
Khung làm việc	**Framework**
Khái niệm	**Concept**

Trình biên dịch	**Compiler**
Trình thông dịch	**Intepreter**
Phân tích từ vựng	**Lexical Analysis**
Phân tích cú pháp	**Syntactical Analysis**
Xử lý ngữ nghĩa	**Semantic Processing**
Kiểu dữ liệu cơ bản	**Primitive Type**
Kiểu dữ liệu cấu trúc	**Structured Type**
Kiểu dữ liệu con trỏ	**Pointer Type**
Mô hình lập trình	**Programming Paradigm**
Lập trình thủ tục	**Procedure Programming**
Điểm vào	**entry point**
Thuộc tính	**Property**
Thủ tục	**Procedure**
Tập trình hướng đối tượng	**Object-Oriented Programming – OOP**
Đối tượng	**Object**
Lớp	**Class**
Kiểu	**Type**
Bao đóng	**Encapsulation**
Phương thức	**Method**
Thừa kế	**Inheritance**
Đa hình	**Polymorphism**
Tính bảo mật	**Security**
Tính tái sử dụng	**Reusablity**
Tính linh hoạt	**Flexibility**
Lập trình khía cạnh	**Aspect-Oriented Programming**
Cấu trúc dữ liệu	**Data Structure**
Danh sách	**List**
Tập hợp	**Set**
Cây	**Tree**
Ánh xạ	**Map**
Môi trường phát triển tích hợp	**Integrated Development Environment – IDE**

Lời tổng kết

Cuốn cẩm nang "Định hướng nghề nghiệp IT" này đã chia sẻ với bạn tổng quan về ngành IT và các nhánh của nó. Bạn cũng đã được tìm hiểu các nghề IT phổ biến nhất trên thị trường việc làm IT. Ngoài ra bạn cũng đã tìm hiểu một số căn bản lập trình.

Mỗi nghề đòi hỏi tính tình, kiến thức và kỹ năng khác nhau cho nên sẽ phù hợp với mỗi loại người khác nhau. Cho dù bạn làm nghề nào trong các nghề đó thì cũng cần nắm được tổng quan hết các nhánh của ngành IT và sự liên quan của chúng trong việc tạo nên 1 hệ thống IT thống nhất.

Trong các nghề đã được chia sẻ, ngoại trừ nghề PM hướng về quản lý thì các nghề còn lại thuần về chuyên môn. Khi bạn mới học xong IT thì hiển nhiên bạn không thể chọn theo nghề PM vì chưa đủ kiến thức và kinh nghiệm để làm quản lý được ngay mà phải chọn nghề về chuyên môn như BA, SD…

Khi bạn đạt đến nấc thang cuối cùng của nghề chuyên môn thì bạn sẽ có 3 hướng:

- Tiếp tục đi sâu về chuyên môn và làm thuần chuyên môn
- Chuyển sang làm quản lý mảng bạn đang làm: ví dụ BA Manager, Engineering Manager, QC Manager, QA Manager, DevOps Manager
- Chuyển sang làm PM

Ngoài ra tôi cũng chia sẻ thêm một số kinh nghiệm của bản thân tôi:

- **ĐAM MÊ**: Khi trẻ chúng ta có nhiều đam mê, chúng ta nên thử hết để xem cái nào chúng ta có khả năng theo lâu dài và nuôi sống được chúng ta thì hãy theo chuyên sâu. Như hồi trẻ tôi đam mê cờ tướng, bóng đá, du lịch và IT. Sau khi đi thi đấu cờ tướng đỉnh cao suốt 16 năm (từ 10 tuổi đến 26 tuổi) thì tôi đã bỏ thi đấu cờ tướng để theo chuyên sâu IT vì nghề cờ tướng không thể nuôi sống tôi. Hiện giờ tôi theo đam mê IT, còn 3 đam mê kia vẫn thỉnh thoảng thực hiện.

- **Mục tiêu**: Ở mỗi giai đoạn sự nghiệp bạn luôn phải có mục tiêu để phấn đấu, đây chính là động lực giúp bạn luôn tiến lên dù gặp khó khăn.

- **Tư duy**: Khi bạn làm bất cứ điều gì thì phải tin vào bản thân mình, vì nếu bạn không tin chính bạn thì cơ thể bạn sẽ chống lại bạn thì hiển nhiên bạn sẽ không thành công, và bạn cũng không thể đòi hỏi người khác tin bạn để giao việc đó cho bạn.
  Ngoài ra khi bạn muốn đạt được điều gì đó (ví dụ trở thành Software Architect hay Project Manager) thì nên gặp gỡ giao lưu với những người đã/đang có những điều đó để học hỏi kiến thức/kinh nghiệm của họ, đây là cách nhanh nhất giúp bạn đạt được điều đó.

- **Học & Hành**: Ngành IT khá rộng lớn nên sau khi bạn học bất cứ thứ gì thì phải tìm cách thực hành để tích lũy kinh nghiệm, từ từ biến thành kỹ năng của bạn. Khi bạn đã có kỹ năng thì sẽ rất khó để quên. Ví dụ kỹ năng lái xe máy, kỹ năng bơi lội…

- **Nghiên cứu**: Trong hệ thống IT thì mọi thứ đều được xây dựng xuất phát từ những khái niệm xoay quanh những luồng nghiệp vụ logic (hợp lý). Vì thế khi tìm hiểu hệ thống IT mới hay công nghệ mới thì phải bắt đầu tìm hiểu từ các khái niệm, sau đó mới đi sâu vô chi tiết công nghệ cụ thể. Sau này công nghệ có thay đổi mà vẫn dựa trên những khái niệm bạn đã biết thì việc nắm bắt công nghệ mới sẽ dễ dàng hơn cho bạn.

- **Phản biện**: Nếu bạn đã học ở Việt Nam từ nhỏ cho đến đại học thì bạn đã bị rơi vào qui trình "Trên bảo dưới phải nghe" mà ít khi dám phản biện (tôi cũng vậy). Tuy nhiên khi đi làm thực tế thì bạn cần thay đổi điều này. Bạn luôn sẵn sàng phản biện lại những gì "trên" nói (giám đốc, trưởng phòng, trưởng nhóm…) nếu bạn cảm thấy những điều đó chưa/không phù hợp với vấn đề hiện tại và đưa ra giải pháp của bạn (hiển nhiên phải kèm theo giải thích thỏa đáng) với mục đích cuối cùng là làm sao có được giải pháp tốt nhất giúp công ty phát triển tốt nhất. Đây là điều những người quản lý cần ở bạn khi bạn làm công ăn lương và cũng là điều mà bạn cần ở nhân viên của bạn khi bạn có công ty riêng!

Hi vọng cuốn cẩm nang này sẽ góp phần giúp bạn có được định hướng tốt nhất cho nghề nghiệp IT của bạn và sẽ trở thành ngôi sao trong lĩnh vực của bạn: **Industry Rockstar** ☺

Mọi ý kiến đóng góp của bạn cho cẩm nang vui lòng gửi về homertruong66@gmail.com nhằm giúp tôi hoàn thiện dần cẩm nang để giúp các bạn IT trẻ định hướng nghề nghiệp IT ngày càng hiệu quả hơn.

Xin cảm ơn các bạn!

Bạn có thắc mắc gì muốn hỏi thêm liên quan đến định hướng nghề nghiệp IT thì hãy kết nối và theo dõi Facebook dành cho mentoring của tôi nhé:

https://www.facebook.com/homertruong66

Chúc các bạn thành công !!!!!!

Homer Truong Le Hoang
http://www.facebook.com/homertruong66
http://www.facebook.com/tech3s.mentor/
http://www.tech3s-mentor.com
http://www.linkedin.com/in/truonglehoang