Talent Enterprise Connecting Hub on 3S System (TECH3S)

CẨM NANG

PHÁT TRIỂN PHẦN MỀM CHUYÊN NGHIỆP

Homer Truong Le Hoang

Vietnam Top Software Professional Mentor

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

Phát triển phần mềm chuyên nghiệp

Lời mở đầu

Trong quyển **Cẩm nang "Định hướng nghề nghiệp IT"**, tôi đã giới thiệu tổng quan ngành IT và các nhánh cũng như các nghề IT phổ biến với Tính tình/Kiến thức/Kỹ năng cần có để theo đuổi từng nghề nhằm giúp các bạn trẻ ĐAM MÊ IT có định hướng nghề nghiệp IT cho mình phù hợp nhất.

Một trong các nghề IT đó là **nhà phát triển phần mềm (Software Developer – SD)** có nhu cầu cao nhất và số lượng nhân lực tham gia cũng đông nhất so với các nghề IT còn lại, bởi vì trong các hệ thống IT ngày nay thì phần mềm đóng vai trò trung tâm.
Tuy nhiên để làm tốt nghề SD này cần rất nhiều kiến thức và kỹ năng để xây dựng ra những phần mềm tốt, hạn chế nhiều lỗi giúp tiết kiệm thời gian và chi phí xây dựng hệ thống IT.

Qua thực tế cho thấy, các bạn trẻ IT Việt Nam khi bắt đầu đi vào nghề SD thường không đủ kiến thức và kỹ năng để tham gia dự án IT thực tế hoặc bối rối không biết sẽ đi theo hướng nào, làm sao để phát triển phần mềm một cách chuyện nghiệp để đem đến hiệu quả công việc của họ tốt nhất và kết quả sản phẩm làm ra cũng tốt nhất.
Vì vậy, thông qua cuốn cẩm nang này, tôi sẽ chia sẻ BÍ QUYẾT để SD phát triển phần mềm một cách chuyên nghiệp, trở thành **nhà phát triển phần mềm chuyên nghiệp** (**Software Professional – SP**) để có thể tham gia vào các đội ngũ phát triển phần mềm chuyên nghiệp quốc tế làm việc tại Việt Nam và các nước trên thế giới.

Nếu bạn đang là SD và muốn trở thành SP thì cuốn cẩm nang này chính xác là kim chỉ nam bạn đang cần. Còn nếu bạn chưa là SD nhưng muốn tìm hiểu về phát triển phần mềm chuyên nghiệp thì cuốn cẩm nang này sẽ giúp bạn định hướng con đường tiến lên SP.
Nếu bạn chưa đọc cuốn Cẩm nang "Định hướng nghề nghiệp IT" của tôi thì hãy tìm đọc trước khi đọc cuốn cẩm nang này nhé.

Trong phần đầu tiên của cuốn cẩm nang này, chúng ta sẽ tìm hiểu các đặc điểm mà SP cần có về mặt con người để theo nghề tốt. Phần hai sẽ nói đến các Kiến thức chuyên ngành giúp SP phát triển phần mềm thành công. Phần cuối sẽ thảo luận các Kỹ năng mềm mà SP cần có để có thể làm việc nhóm hoặc làm việc độc lập một cách hiệu quả nhất.

Nào chúng ta bắt đầu…

I. Con người

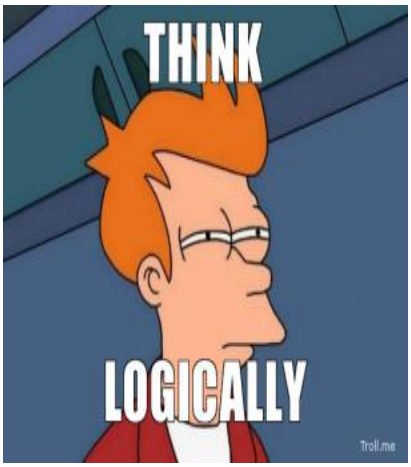
1) Tư duy (Attitude)

Khi bạn làm một việc gì đó mà lúc nào cũng nghĩ việc này khó quá, sợ mình làm không được thì con người bạn sẽ sinh ra một lực vô hình chống lại hành động của bạn và khả năng cao là bạn sẽ thất bại với việc đó.

Ngược lại, dù việc rất khó (nhưng trong khả năng của bạn chứ không thuộc dạng viễn vông) nhưng lúc nào bạn cũng tin tưởng là mình sẽ làm được và luôn cố gắng hết sức cũng như tâm trí để thực hiện thì cơ thể bạn sẽ sản sinh ra một nguồn năng lượng tích cực sẽ giúp bạn tăng khả năng hoàn thành công việc đáng kinh ngạc. Đó là bạn đã kết hợp giữa sức mạnh thể chất/trí tuệ với sức mạnh tinh thần một cách hài hòa.

Cho nên, **tư duy "Tôi có thể" ("Can do" attitude)** cực kỳ quan trọng đóng góp vào sự thành công của bạn ngay từ khi bạn chuẩn bị/bắt đầu một việc gì đó.

Trong thế giới quanh ta, mọi vấn đề trong mọi lĩnh vực luôn vận hành theo những **tính hợp lý (logic)** nào đó. Phần mềm giúp giải quyết những vấn đề này một cách tự động hóa thì cũng phải tuân theo những logic đó, vì thế người viết ra phần mềm phải luôn **suy nghĩ logic (logial thinking)** thì mới tạo ra được những phần mềm tốt, né những lỗi sai theo kiểu phi logic.



Mỗi khi bạn bắt đầu tìm hiểu một lĩnh vực mới nào đó, ngoài việc học qua trường lớp, qua sách vở, và tự học thì bạn nên tìm **môi trường nơi mà những người khác đã thành công** trong lĩnh vực này như các hội/nhóm để bạn có cơ hội học hỏi kiến thức/kinh nghiệm từ họ để ngày nào đó bạn sẽ thành công như họ.

Tốt hơn nữa là bạn có được **Mentors** định hướng cho bạn đi và theo bạn giúp bạn sửa sai khi bạn học/làm việc thực tế. Trong lĩnh vực phát triển phần mềm thì mentor gần bạn nhất chính là **trưởng nhóm/đồng nghiệp**, tiếp theo là **những bạn trong mạng lưới bạn bè của bạn** hay những người khác trong lĩnh vực này mà bạn gặp trong các sự kiện mà bạn tham dự.

MENTOR follows and helps you a long the way.

Các công nghệ phục vụ phát triển phần mềm luôn phát triển rất nhanh. Cho nên bạn phải có **tinh thần ham học hỏi (Eager-to-learn Spirit)** và **tính thích ứng cao (High Adaptability)** để học hỏi rồi nắm bắt công nghệ một cách kịp thời thì mới làm ra những sản phẩm tốt nhất.

Là một SP thì bạn luôn phải có **tinh thần làm chủ (Enterpreneurial Spirit)** của sản phẩm bạn tham gia phát triển cho dù bạn đang làm thuê lãnh lương. Bởi vì khi có tinh thần làm chủ thì bạn sẽ nhìn sản phẩm một cách bao quát hơn, tìm hiểu chi tiết hơn, luôn luôn tìm cách tốt nhất để xây dựng sản phẩm. Và khi sản phẩm tốt giúp công ty phát triển tốt lên thì bạn cũng được hưởng lợi cả về tinh thần và vật chất.

Ngược lại bạn nghĩ bạn làm công ăn lương thì chỉ làm cho xong những việc được giao rồi về thì bạn sẽ khó tiến lên cao trên con đường sự nghiệp và sản phẩm công ty cũng không được phát triển hết tiềm năng của nó.

2) ĐAM MÊ (PASSION)

Bạn hãy để ý khi bạn làm việc gì đó một cách ĐAM MÊ thì sẽ có 1 nguồn năng lượng được sinh ra chạy khắp cơ thể bạn giúp bạn hứng thú với việc bạn đang làm và tập trung hoàn toàn vào đó, đôi khi quên cả thời gian cũng nhưng những sự kiện khác đang diễn ra quanh bạn; và kết quả đạt được thường rất tốt đẹp.

Ngược lại bạn làm việc gì đó bạn không đam mê mà do bị ép hay vì lý do gì đó không thể không làm thì lý trí của bạn không ủng hộ cho bạn hoàn thành công việc này và hầu hết kết quả sau cùng là tạm tạm hoặc không đạt yêu cầu.

Bởi vậy, **ĐAM MÊ đóng vai trò thiết yếu trong việc thành công ở bất cứ lĩnh vực nào**, nhất là mảng phát triển phần mềm đầy thách thức và áp lực dù không kém phần thú vị. Cho nên, bạn hãy xem xét lại bạn có thực sự ĐAM MÊ công việc phát triển phần mềm hay không?
Nếu câu trả lời là có thì xin chúc mừng bạn!
Nếu bạn chưa có câu trả lời ngay thì hãy tạo cơ hội cho bạn thử sức với nghề này.
Nếu bạn đã làm nghề này một thời gian đủ dài mà vẫn cảm thấy không có được sự đam mê thì bạn nên tìm ĐAM MÊ của bạn trong các cơ hội nghề nghiệp khác.

3) Nền tảng (Background)

Một ngôi nhà vững chãi thì chắc chắn cái móng của nó phải vững vàng.

Một cái cây khỏe thì chắc chắn gốc rễ của nó rất tốt.

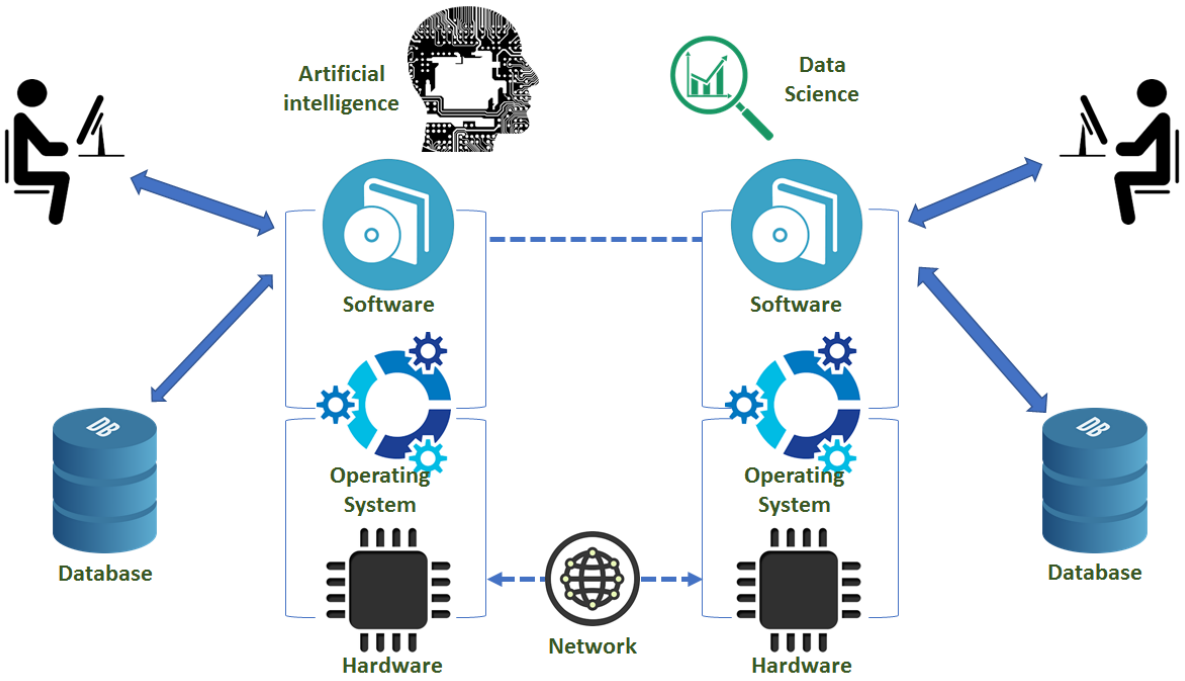
Tương tự, khi làm bất cứ việc gì thì chúng ta phải quan tâm xây dựng nền tảng vững chắc trước, từ đó mới phát triển tiếp lên thì sẽ tốt hơn.

Ví dụ: thay vì chúng ta nhảy vô học ngay một **ngôn ngữ lập trình (Programming Language) – NNLT** cụ thể nào đó thì chúng ta nên tìm hiểu trước căn bản của NNLT, chương trình dịch và thực hành bằng một NNLT cụ thể để hiểu rõ các đặc điểm, tính chất của NNLT. Sau này chúng ta chuyển qua học các NNLT khác sẽ nhanh hơn nhiều so với chúng ta học từng NNLT riêng rẽ vì chúng ta đã nắm được nền tảng của NNLT.

II. Kiến thức chuyên ngành

1) Nền tảng IT (IT Foundation)

Phần mềm là 1 nhánh của ngành IT cũng là trung tâm của hệ thống IT, nên để xây dựng phần mềm tốt thì ta phải nắm vững nền tảng IT. Phần tổng quan ngành IT tôi đã chia sẻ trong cuốn cẩm nang "Định hướng nghề nghiệp IT", giờ chúng ta xem lại bức tranh toàn cảnh 1 hệ thống IT cơ bản và đi sâu hơn tí ở mỗi nhánh về khía cạnh hỗ trợ việc phát triển phần mềm.



- Người dùng tương tác với **Máy Vi Tính (Computer) – MVT** thông qua **Phần mềm (Software)**.

- Phần mềm nhờ **Hệ Điều Hành (Operating System) – HĐH** để điều khiển **Phần cứng (Hardware)** thực hiện lệnh của người dùng. Như vậy khi phát triển phần mềm chúng ta phải quan tâm đến loại HĐH mà phần mềm sẽ chạy trên đó để chọn công nghệ phát triển phù hợp. Ví dụ: chọn công nghệ .Net thì phần mềm chỉ chạy được trên Windows, chọn công nghệ Java thì phần mềm chạy được trên nhiều HĐH khác nhau…

  Đối với các phần mềm chuyên dùng để điều khiển các thiết bị phần cứng thì ngoài kiến thức viết phần mềm chuyên dụng này thì ta phải tìm hiểu cách thức hoạt động của các thiết bị phần cứng đó.

- Phần mềm cần tương tác với các phần mềm khác hoặc bản thân phần mềm được triển khai trên nhiều MVT (ví dụ: triển khai mô đun cho giao diện người dùng, mô đun xử lý nghiệp vụ) thì phải thông qua **Mạng Máy Tính (MMT)**.

  Tùy theo yêu cầu phần mềm mà ta chọn loại MMT phù hợp. Ví dụ phần mềm chỉ xài trong nội bộ 1 công ty thì chúng ta chỉ cần triển khai phần mềm trên MMT là **Mạng nội bộ (Local Area Network – LAN)**; còn phần mềm được người dùng khắp nơi trên thế giới xài thì chúng ta triển khai trên MMT là **Mạng toàn cầu (Internet)** với **giải pháp trong nhà (In-house Solution)** hoặc dùng **giải pháp đám mây (Cloud Solution)**.

- Dữ liệu người dùng được lưu trữ dưới **Cơ Sở Dữ Liệu (Database) – CSDL.**

  Tùy yêu cầu phần mềm mà ta chọn 1 trong 2 loại CSDL: CSDL quan hệ (Relational Database) hoặc CSDL NoSQL.

  Ví dụ: phần mềm cho các ngân hàng thì các đối tượng mà phần mềm xử lý (như Tài khoản, Người gửi tiết kiệm, Người vay, Người cho vay, Thẻ ATM, Thẻ tín dụng…) có quan hệ với nhau khá phức tạp (dẫn đến truy xuất chậm vì nhu cầu kết nối dữ liệu để lấy dữ liệu tổng hợp) và dữ liệu không lớn nhưng đòi hỏi **tính toàn vẹn (integrity)** rất cao, vì vậy SQL là lựa chọn tốt hơn NoSQL.

  Ngược lại, phần mềm quản lý sản phẩm xuất nhập kho thì các đối tượng mà phần mềm xử lý (Sản phẩm, Kho) không có quan hệ phức tạp và dữ liệu khá lớn thì NoSQL là lựa chọn tốt hơn SQL.

- **Trí tuệ nhận tạo (Artificial Intellience)** và **Khoa học dữ liệu (Data Science)** là 2 mảng chuyên biệt. Khi nào bạn đụng đến các dự án phần mềm liên quan đến 2 mảng này thì sẽ bỏ thời gian ra tìm hiểu kỹ **kiến thức lĩnh vực (Domain Knowledge)** thì mới bắt đầu viết phần mềm trên đó được.

Vậy thì để phát triển phần mềm chuyên nghiệp thì bạn phải nắm vững **Công Nghệ Phần Mềm (CNPM)** và có thêm kiến thức tối thiểu của 4 nhánh (đã được đặc tả trong cuốn Cẩm nang "Định hướng nghề nghiệp IT"):

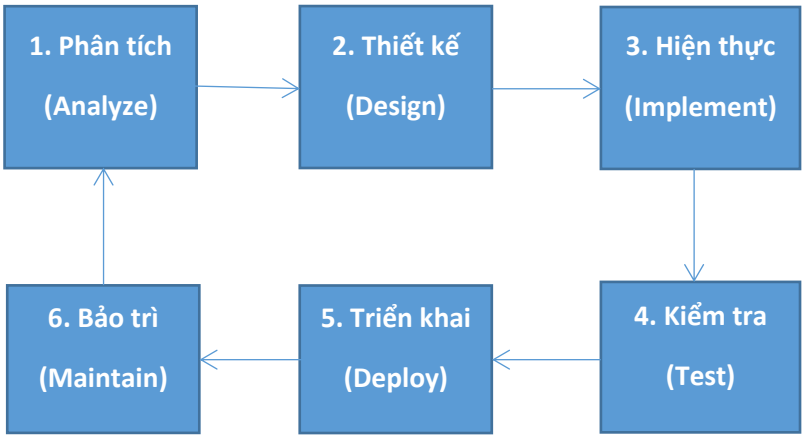
- Hệ điều hành (Operating System)
- Phần cứng (Hardware) hay có thể gọi là **Cấu trúc máy tính (Computer Structure)**
- Cơ sở dữ liệu (Database)
- Mạng máy tính (Computer Network)

Chúng ta đi vào tìm hiểu CNPM…

2) Công nghệ phần mềm (Software Engineering)

Để phát triển phần mềm tốt thì hiển nhiên chúng ta phải biết cách thức xây dựng phần mềm hay còn gọi bằng từ chuyên môn hơn là **CNPM**. Thế thì CNPM là gì vậy?

CNPM định nghĩa theo cách hàn lâm thì hơi dài dòng khó hiểu. CNPM định nghĩa nôm na dễ hiểu hơn là cách thức để chúng ta xây dựng 1 phần mềm theo 6 bước sau đây:



Đây cũng được gọi là **vòng đời phát triển phần mềm (Software Development Life Cycle – SDLC)**.

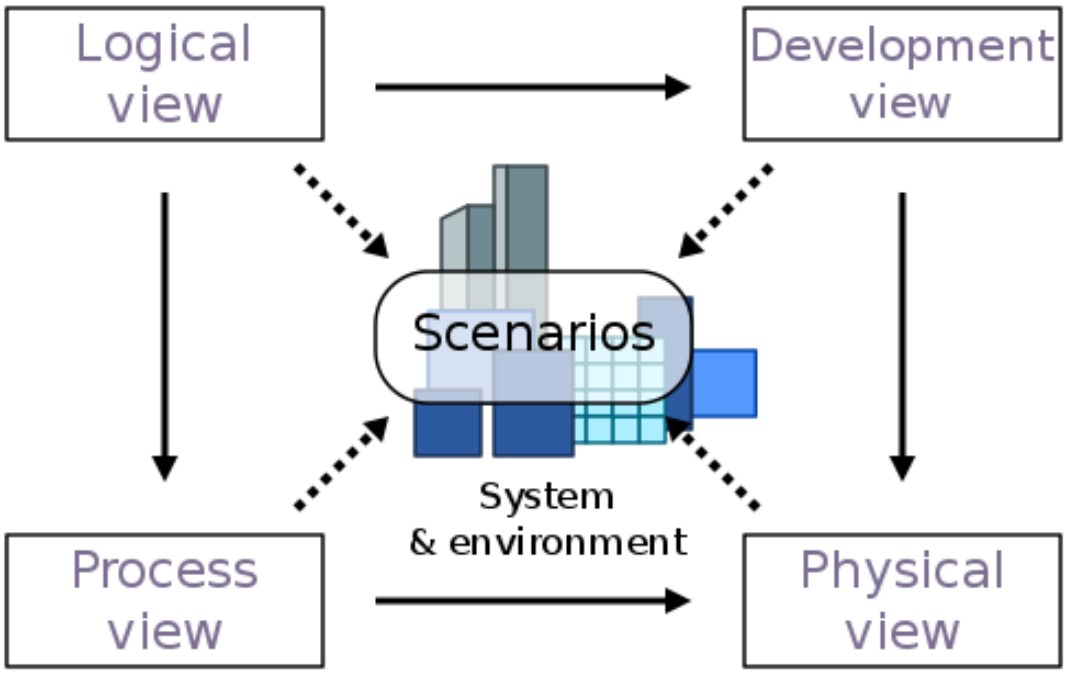
Từ thời CNPM mới ra đời thì phương pháp phát triển phần mềm là **phương pháp thác nước (Waterfall Methodology),** theo đó tất cả 6 bước trên được thực hiện tuần tự TRÊN TẤT CẢ TÍNH NĂNG phần mềm. Các phần mềm lúc trước có các **yêu cầu chức năng (Functional Requirement – FR)** biết trước là sẽ đáp ứng đúng nhu cầu của người dùng, ví dụ như phần mềm kế toán, phần mềm quản lý nhân sự… nên phương pháp thác nước áp dụng xây dựng 1 phần mềm như vậy mất 6 tháng đến 1 năm mới ra mắt người dùng cũng được.
Tuy nhiên, sau này người ta thấy phát triển cả đống tính năng phần mềm 1 lèo suốt 6 tháng đến 1 năm rồi mới đưa người dùng xài thì lỡ có tính năng gì đó người dùng không thật sự muốn hoặc muốn chỉnh sửa gì đó thì chi phí thay đổi rất cao. Cho nên phương pháp xây dựng phần mềm mới đã ra đời: đó là **phương pháp nhanh & lặp (Agile Methodology)**. Với phương pháp này thì 6 bước trên được thực hiện tuần tự TRÊN VÀI TÍNH NĂNG phần mềm trong vài tuần, sau đó được lặp lại cho những tính năng tiếp theo.

Một hiện thực của Agile Methodology rất phổ biến là SCRUM. Vì vậy trong cuốn cẩm nang này, SCRUM sẽ được dùng chính cho phương pháp phát triển phần mềm.

Chúng ta bắt đầu tìm hiểu chi tiết từng bước của hình trên.

- **Phân tích (Analyze)**: Các FR của phần mềm được **Business Analyst (BA)** phân tích để làm rõ các **luồng nghiệp vụ (Business Flow)**, các **qui luật nghiệp vụ (Business Rule)**, các **cách người dùng tương tác với phần mềm (User Story) – US**, chuyển đổi giao diện (UI Transition) trong các USs… Phần phân tích này tuy được BA làm và chịu trách nhiệm chính nhưng SD cũng phải tham gia nắm bắt rõ thì khi vào phần Thiết kế & Hiện thực sẽ hạn chế lỗi liên quan đến FR.

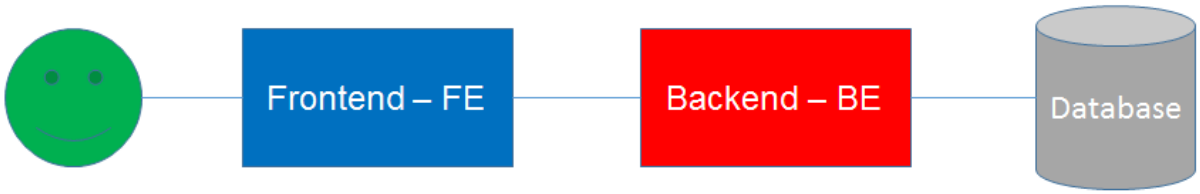
- **Thiết kế (Design)**: Dựa trên kết quả phân tích các FR, **Kiến trúc sư phần mềm (Software Architect – SA)** cùng các thành viên SDs, DevOps khác trong nhóm **Architecture Review Board (ARB)** sẽ thiết kế hệ thống IT (bao gồm phần mềm). Tùy theo mô hình phát triển phần mềm được sử dụng mà các kết quả (các **lược đồ thiết kế - Design Diagram**) sẽ khác nhau. Mô hình được sử dụng thông dụng nhất là **RUP 4+1**:
  - ***Logical View*:** Thể hiện các tính năng luận lý của phần mềm.
  - ***Development View*:** Hiển thị góc nhìn về phát triển phần mềm (Kiến trúc phần mềm, cách hệ thống IT tích hợp với các hệ thống bên ngoài…)
  - ***Process View*:** Thể hiện các qui trình nghiệp vụ của phần mềm.
  - ***Physical View*:** Hiện thị các thành phần của hệ thống IT được triển khai trên các máy chủ như thế nào.
  - ***Scenarios*:** Các góc nhìn trên dựa trên một tập hợp các tình huống tương tác người dùng, tương tác giữa các đối tượng khác nhau trong hệ thống IT.



- **Hiện thực (Implement)**: Bạn SA dựa trên các FR và các **yêu cầu phi chức năng (Non-Functional Requirement – NFR)** để lựa chọn 1 **nhóm công nghệ (Technology Stack)** phù hợp, rồi lập trình xây dựng 1 **khung hệ thống (System Skeleton).** Tiếp đó, đội ngũ SDs sử dụng khung hệ thống này và dựa trên phần phân tích các FR và các lược đồ thiết kế để lập trình xây dựng các **tính năng phần mềm (Feature)**. Theo SCRUM, mỗi Feature gồm nhiều USs sẽ được hiện thực trong một/vài **lần chạy nước rút (Sprint)**. Một Sprint thường diễn ra từ 2-4 tuần tùy theo giai đoạn của dự án.

  Mỗi US có thể bao gồm một hoặc nhiều **giao diện người dùng (User Interface – UI)** và sự chuyển đổi giữa các UIs trong quá trình người dùng tương tác với phần mềm. Vì vậy hiện thực 1 US sẽ bao gồm 2 phần:

  - **Lập trình phía người dùng (Frontend Programming – FE):** FE có thể là Web hoặc Mobile App, các UIs được chuyển đổi theo 1 thứ tự được định nghĩa trước, dữ liệu thể hiện trên UIs được lấy từ BE.
  - **Lập trình xử lý nghiệp vụ phần mềm (Backend Programming – BE):** xử lý toàn bộ nghiệp vụ phần mềm, lưu dữ liệu vào CSDL, cung cấp dữ liệu cho FE.



  Một cách trực quan chúng ta thấy phần BE phức tạp hơn phần FE. Do đó, SD làm BE sẽ đòi hỏi nhiều kiến thức/kinh nghiệm/kỹ năng hơn SD làm FE, hiển nhiên SD học để làm BE cũng sẽ mất nhiều thời gian hơn FE SD và lương trung bình cũng sẽ cao hơn FE SD.

- **Kiểm tra (Test)**: Ngoài **Quality Control (QC)** và BA kiểm tra chất lượng các Feature đã được hoàn tất thì SD trong lúc hiện thực các tính năng cũng thực hiện song song việc **kiểm tra đơn vị (Unit Test – UT)**, trong đó SD viết thêm các đoạn code bên ngoài để kiểm tra từng **mô đun (module)** của hệ thống xem có chạy đúng khi nó chạy một mình (không liên kết với các mô đun khác) hay không. Việc hiện thực UT có giá trị rất lớn trong quá trình bảo trì phần mềm vì UT giúp phát hiện những thay đổi của SD khi thực hiện các tính năng sau có làm hỏng những tính năng đã được hiện thực trước không.
- **Triển khai (Deploy):** Khi các USs trong Sprints đã được hiện thực và kiểm tra ok hết rồi thì DevOps sẽ triển khai liên tiếp cho người dùng xài.
- **Bảo trì (Maintain):** Sau khi người dùng đã xài phần mềm thì SCRUM Team bắt đầu làm 2 việc song song: phát triển USs mới & bảo trì hệ thống IT.

Như vậy đến đây chúng ta thấy được vai trò của SA & SD:

- SA phối hợp với BA phân tích FR
- SA phân tích FR & NFR rồi thiết kế và chọn Technology Stack cho hệ thống IT
- SA thiết kế và xây dựng System Skeleton
- SD đọc hiểu các USs được giao
- SD hiện thực các USs (kèm UTs nếu có yêu cầu) trên các công nghệ SA đã chọn:
  - FE SD lập trình xây dựng và chuyển đổi UIs, lấy dữ liệu từ BE.
  - BE SD lập trình xử lý nghiệp vụ của phần mềm, tương tác với CSDL để lưu/đọc dữ liệu cung cấp cho FE.

Ví dụ: xây dựng 1 phần mềm thương mại điện tử (e-commerce software).

- SA phối hợp với BA phân tích yêu cầu phần mềm:
  - Feature 1 : Quản lý sản phẩm (Product)
    - US 1: Nhập sản phẩm vô hệ thống
    - US 2: Tìm các sản phẩm trên website
    - US 3: Chọn sản phẩm bỏ vô giỏ hàng
  - Feature 2 : Quản lý đơn hàng (Order)
    - US 1: Tạo đơn hàng
    - US 2: Thanh toán đơn hàng
    - US 3: Xem danh sách đơn hàng
- SA thiết kế và chọn Technology Stack cho hệ thống IT
  - Frontend Web: HTML/CSS/Javascript, SASS, Bootstrap, React, Redux, Saga.
  - Backend: Java, Spring REST, Spring Security, JWT, Hibernate, MySQL, Redis.
- SA thiết kế và xây dựng System Skeleton: đầu ra là các lược đồ thiết kế và **bộ mã nguồn (Source Code)** lưu trong SVN or Git.
- SD đọc hiểu các USs được giao
- SD hiện thực các USs (kèm UTs nếu có thời gian) trên Technology Stack SA đã chọn:
  - FE SD lập trình xây dựng các UIs và chuyển đổi UIs: UI Tìm sản phẩm → UI Kết quả tìm sản phẩm → UI xem chi tiết 1 sản phẩm và bỏ vô giỏ hàng → UI xem đơn hàng được tạo ra → UI thanh toán → UI cảm ơn sau khi thanh toán thành công.
  - BE SD lập trình xử lý nghiệp vụ của phần mềm: Khi người dùng tìm sản phẩm thì trả về các sản phẩm hợp lệ (còn trong kho, sản phẩm bán được…); khi người dùng xem chi tiết 1 sản phẩm thì trả về thông tin của sản phẩm đó; khi người dùng thanh toán 1 đơn hàng thì hệ thống thực hiện thanh toán và cập nhật sản phẩm còn lại trong kho…

Trong công việc phát triển phần mềm hàng ngày, bạn làm SD sẽ giao tiếp nhiều với các bạn làm BA, SA, DevOps, và QC. Vì thế, việc bạn hiểu được công việc của các nghề này sẽ giúp cho việc trao đổi giữa bạn với những người làm trong nghề này hiệu quả hơn rất nhiều.

Nếu có cơ hội thì bạn nên làm một phần việc của những nghề trên, điều này sẽ giúp bạn nắm rõ hơn hệ thống IT bạn đang làm việc trên đó. Đừng bao giờ tự giới hạn khả năng của mình trong nghề mình đang làm. Tuy nhiên, phải làm tốt công việc chính của mình, sau đó mới tìm hiểu những công việc liên quan.

Trong các **sản phẩm công nghệ (Tech Product)** tôi đã tham gia làm lúc tôi ở bên Canada, tôi làm công việc chính là SD. Khi những lúc tôi đã hoàn thành xong việc của tôi và những vị trí khác như BA, DevOps, QC việc hơi nhiều thì tôi cũng nhảy vô vừa học vừa làm phụ họ. Nhờ đó từ từ tôi tích lũy được kiến thức/kinh nghiệm/kỹ năng trong các nghề đó.

Qua mỗi dự án phần mềm, ngoài kiến thức lĩnh vực tôi thu thập được thì tôi cũng tìm hiểu rõ mọi ngóc ngách của Technology Stack mà SA đã chọn từ lý thuyết đến cách mỗi công nghệ được áp dụng vô dự án thực tế. Khi có gì không hiểu thì tôi hỏi SA, đây là cách nhanh nhất để tôi học cách thức xây dựng kiến trúc phần mềm thực tế (lý thuyết về kiến trúc phần mềm rất nhiều, đọc mà không thực hành vào dự án thực tế thì cũng không hiểu và không nhớ được) cũng như hiểu hơn về công việc của vị trí SA.

Sau khi làm qua một số Tech Products, trong thời gian rãnh tôi đã tự mình xây dựng 1 Tech Product từ A-Z với nghiệp vụ là nghiệp vụ của 1 trong các Tech Product tôi đã làm và Technology Stack là tổng hợp các Technology Stack của các Tech Products đó. Trên Tech Product này, tôi đã làm qua toàn bộ qui trình 6 bước của CNPM và làm tất cả các vai trò của SCRUM Team. Điều này đã giúp tôi nắm rất vững CNPM.

Trong SDLC, ngoài việc hỗ trợ phân tích với BA thì công việc chính của SD là lập trình FE hoặc BE hoặc tốt nhất là lập trình cả FE/BE. Phần tiếp theo chúng ta sẽ tìm hiểu về chúng.

3) Lập trình Frontend & Backend (Frontend/Backend Programming)

Từ khi công nghệ Web ra đời, hầu hết các phần mềm đều dựa trên nền Web. Trước khi các Javascript Frameworks phổ biến xuất hiện thì các trang Web trong 1 **ứng dụng Web (Web Application,** hay còn gọi là **Web App** cho gọn**)** được xử lý chủ yếu bên máy chủ bởi các công nghệ như CGI (Common Gateway Interface), ASP (Active Server Page), JSP (Java Server Page)… Các ứng dụng này được gọi là **ứng dụng đa trang (Multi-Page Application – MPA)**.

Nhược điểm của MPA là trình duyệt tải xong 1 trang Web hoàn chỉnh (gồm cấu trúc trang Web và dữ liệu của ứng dụng) khá mất thời gian, nhất là trang Web có cấu trúc phức tạp và nhiều dữ liệu. Tồi tệ hơn khi mạng bị chậm vì lý do gì đó thì sẽ bị "time out", trang Web không được lấy về làm người dùng thất vọng với Web App, điều này ảnh hưởng đến **trải nghiệm người dùng (User Experience – UX)** rất lớn.
Ngoài ra việc mất thời gian để lấy về các trang Web khiến cho việc tương tác trên UI của người dùng không được mượt mà như là họ đang tương tác trên **ứng dụng Windows (Windows App)** vốn chạy cục bộ trên máy người dùng.

Windows App thì bị hạn chế là người dùng phải cài đặt thêm phần mềm liên quan đến công nghệ xây dựng cái Windows App thì mới xài được nó. Ví dụ xài công nghệ Java thì phải cài JRE (Java Runtime Environment) mới chạy được 1 Java Windows App. Trong khi xài Web App thì chỉ cần trình duyệt, vốn đã có sẵn trong bất cứ HĐH nào hiện nay.

Để giải quyết cho hạn chế của Windows App và nhược điểm của MPA thì **ứng dụng đơn trang (Single Page Application – SPA)** đã ra đời. SPA cũng là một Web App như MPA, nhưng khác ở chỗ cơ bản là SPA chỉ có 1 trang Web duy nhất.
Khi người dùng bắt đầu vô phần mềm thì trình duyệt chỉ tải 1 trang Web 1 lần duy nhất (có thể mất thời gian chút nhưng chỉ lần đầu, những lần sau vô lại sẽ nhanh hơn nhiều do trình duyệt đã lưu lại cấu trúc trang Web, chỉ cập nhật những thành phần bị thay đổi), những tương tác của người dùng trên các UI được thực hiện ở trong nội bộ trình duyệt chứ trình duyệt không tải trang Web nào khác như cách MPA làm.

Như vậy, cấu trúc SPA sẽ bao gồm 1 trang Web duy nhất, trang này sẽ bao gồm 1 bộ UIs và giữa các UIs sẽ có sự liên kết với nhau, y như sự liên kết của các trang Web, để người dùng có thể tương tác trên các UIs đó và chuyển đổi giữa chúng. Dữ liệu để hiển thị trên UIs được lấy từ máy chủ.

Từ đây nghề lập trình FE ra đời để xây dựng nên bộ UIs, quản lý cách thức chuyển đổi giữa chúng, hiển thị dữ liệu lên UIs. Thế thì dữ liệu của ứng dụng lấy từ đâu ra? Đây là việc của các bạn lập trình Backend lo về xử lý nghiệp vụ ứng dụng và lưu/đọc dữ liệu cho FE hiển thị để người dùng tương tác. Dữ liệu được đưa từ Frontend qua Backend và trở về dựa trên công nghệ AJAX (Asynchronous JavaScript And XML) đã khá phổ biến trong các trình duyệt.

FE như đã nói ở trên liên quan đến Web. Tuy nhiên chúng ta có thể mở rộng mô hình này cho các **thiết bị di động (Mobile Device)**, **điện thoại thông minh (Smart Phone)**.

Đến đây thì chúng ta thấy lập trình FE sẽ bao gồm cả Web UIs & Mobile UIs. Nghĩa là chúng ta sẽ có FE Web & FE Mobile. Thế nhưng do hiện nay ứng dụng Web chiếm đa số trên thị trường nên khi nghe đến FE thì mọi người tự hiểu là đang nói đến FE Web, còn Mobile thì hiểu là FE Mobile.

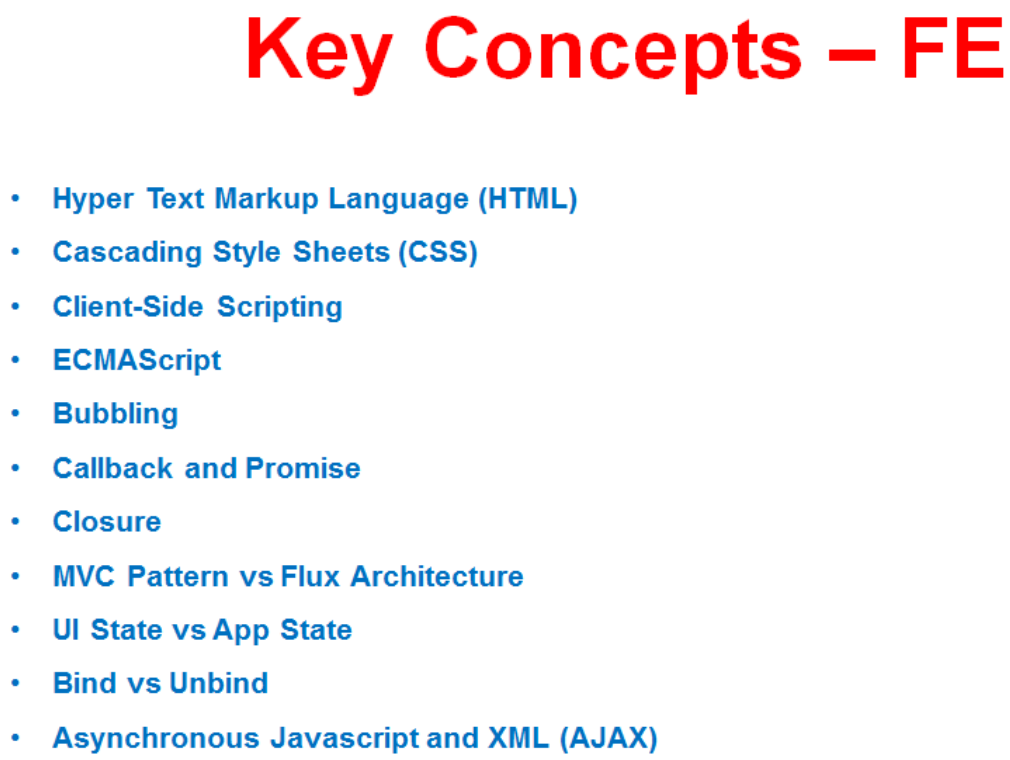
Còn lập trình BE sẽ gắn với 1 công nghệ nào đó, ví dụ: Java Backend Programing, hay họ có thể bỏ chữ "Backend" để viết ngắn gọn hơn là Java Programming.

Kể từ khi lập trình FE ra đời đến nay thì số lượng FE Frameworks (cả Web & Mobile) tăng khá nhanh làm cho việc nắm bắt các công nghệ FE trở nên khó hơn.

Vì vậy nghề Software Developer đã được phân nhánh thành **FE Developer** & **BE Developer**. Nếu bạn giỏi có thể làm cả 2 nhánh thì gọi là **Fullstack Developer**.

Như vậy, để trở thành 1 Software Professional thì bạn có thể chọn làm FE Developer hoặc BE Developer. **Tốt nhất, bạn hãy đặt mục tiêu cao hơn là phấn đấu trở thành Fullstack Developer.**

Để làm FE Web tốt, bạn cần nắm các khái niệm chính sau:



Khi nắm vững các khái niệm này rồi thì khi vào dự án, bạn sẽ nắm bắt các công nghệ FE nhanh hơn bởi vì các công nghệ luôn dựa trên các khái niệm.

Để làm BE tốt, bạn cần nắm được một số **mẫu thiết kế (Design Pattern) – DP** thông dụng. Dưới đây là toàn bộ các DP được chia thành 3 nhóm.

Creational Patterns

Builder	Separates object construction from its representation
Factory Method	Creates an instance of several derived classes
Prototype	A fully initialized instance to be copied or cloned
Singleton	A class of which only a single instance can exist

Structural Patterns

Adapter	Matches interfaces of different classes
Bridge	Separates an object’s interface from its implementation
Composite	A tree structure of simple and composite objects
Decorator	Adds responsibilities to objects dynamically
Facade	A single class that represents an entire subsystem
Proxy	An object representing another object

Behavioral Patterns

Chain of Resp.	A way of passing a request between a chain of objects
Command	Encapsulates a command request as an object
Iterator	Sequentially accesses the elements of a collection
Mediator	Defines simplified communication between classes
Memento	Captures and restores an object's internal state
Observer	A way of notifying change to a number of classes
State	Alters an object's behavior when its state changes
Strategy	Encapsulates an algorithm inside a class
Template Method	Defers the exact steps of an algorithm to a subclass
Visitor	Defines a new operation to a class without change

Ngoài ra BE Developer cần nắm rõ các khái niệm chủ yếu sau đây để thiết kế và hiện thực phần mềm tối ưu nhất. Những khái niệm này phổ biến nên bạn có thể tìm đọc trên internet và sau đó áp dụng vô dự án để hiểu rõ và nhớ lâu.

Key Concepts – BE

- Static Web Page
- Dynamic Web Page
- Web Page Template
- Web Service
- Web API
- Web Filter
- Interceptor
- Multi-threading
- Model View Controller (MVC)
- Data Scope

- Application Component
- Inversion of Control (IoC)
- Dependency Injection (DI)
- Business Transaction
- Database Transaction
- Database Connectivity
- Connection Pooling
- Object/Relational Mapping (ORM)
- Lazy/Eager Loading
- Database Management System (DBMS)

- Design Patterns
- Exception Handling
- Logging
- Caching
- System Security
- Concurrent Requests
- Database Concurrency
- Database Partitioning
- Asynchronous Processing
- Internationalization
- Application Availability

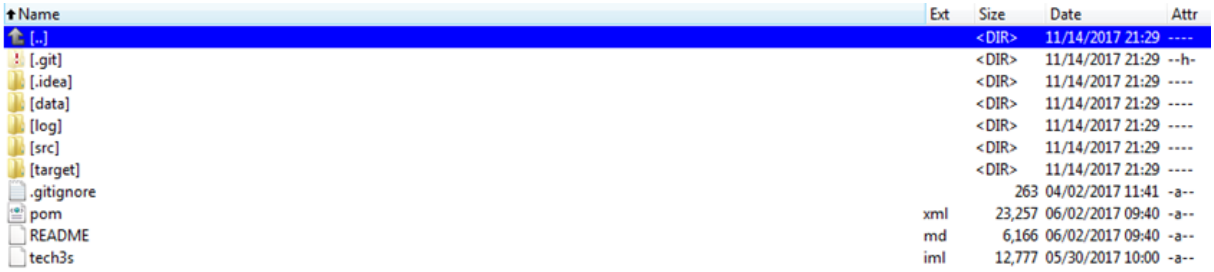
- Load Balancing
- Application Scalability
- Application Monitoring
- Verification
- Validation
- Unit Test
- System Test
- Integration Test
- Regression Test
- Monolithic Application
- Micro-service Application

Khi lập trình thì SD có thể tạo files mới hoặc chỉnh sửa files đã tồn tại trên bộ mã nguồn. Giả sử sau khi chỉnh sửa 1 file N lần, ta muốn biết ở lần thứ 3 ta đã sửa những gì thì phải làm sao?

Trong quá trình phát triển phần mềm, các SDs sẽ lập trình trên 1 cây thư mục gồm thư mục gốc của dự án chứa các thư mục con và files. Tất cả thư mục và files trong thư mục gốc này đều được lưu toàn bộ quá trình sửa đổi bởi 1 **Hệ quản lý phiên bản nào (Version Control System – VCS))** đó.

Các VCS phổ biến khi xưa là **ClearCase, SVN**. Gần đây một chuẩn mới xuất hiện là **Git** trở nên rất phổ biến. Các hiện thực của chuẩn Git khá phổ biến là **GitHub và BitBucket**.

Đây là 1 ví dụ của 1 dự án phần mềm được quản lý bởi GitHub:



Name	Ext	Size	Date	Attr
[..]		<DIR>	11/14/2017 21:29	----
[.git]		<DIR>	11/14/2017 21:29	--h-
[.idea]		<DIR>	11/14/2017 21:29	----
[data]		<DIR>	11/14/2017 21:29	----
[log]		<DIR>	11/14/2017 21:29	----
[src]		<DIR>	11/14/2017 21:29	----
[target]		<DIR>	11/14/2017 21:29	----
.gitignore		263	04/02/2017 11:41	-a--
pom	xml	23,257	06/02/2017 09:40	-a--
README	md	6,166	06/02/2017 09:40	-a--
tech3s	iml	12,777	05/30/2017 10:00	-a--

Thư mục gốc có chứ 1 thư mục ẩn là ".git" để lưu trữ toàn bộ quá trình sửa đổi các thư mục & files nằm bên dưới thư mục gốc này.

Nắm vững và thao tác thuần thục trên VCS như Git là điều bắt buộc cho tất cả SDs, bạn có thể học chuẩn Git ở đây khá dễ hiểu: https://www.atlassian.com/git/tutorials.

Học phải đi đôi với hành, nên vừa học với trang ở trên thì bạn hãy tạo 1 dự án để thực hành hoàn toàn miễn phí ở đây: https://github.com.

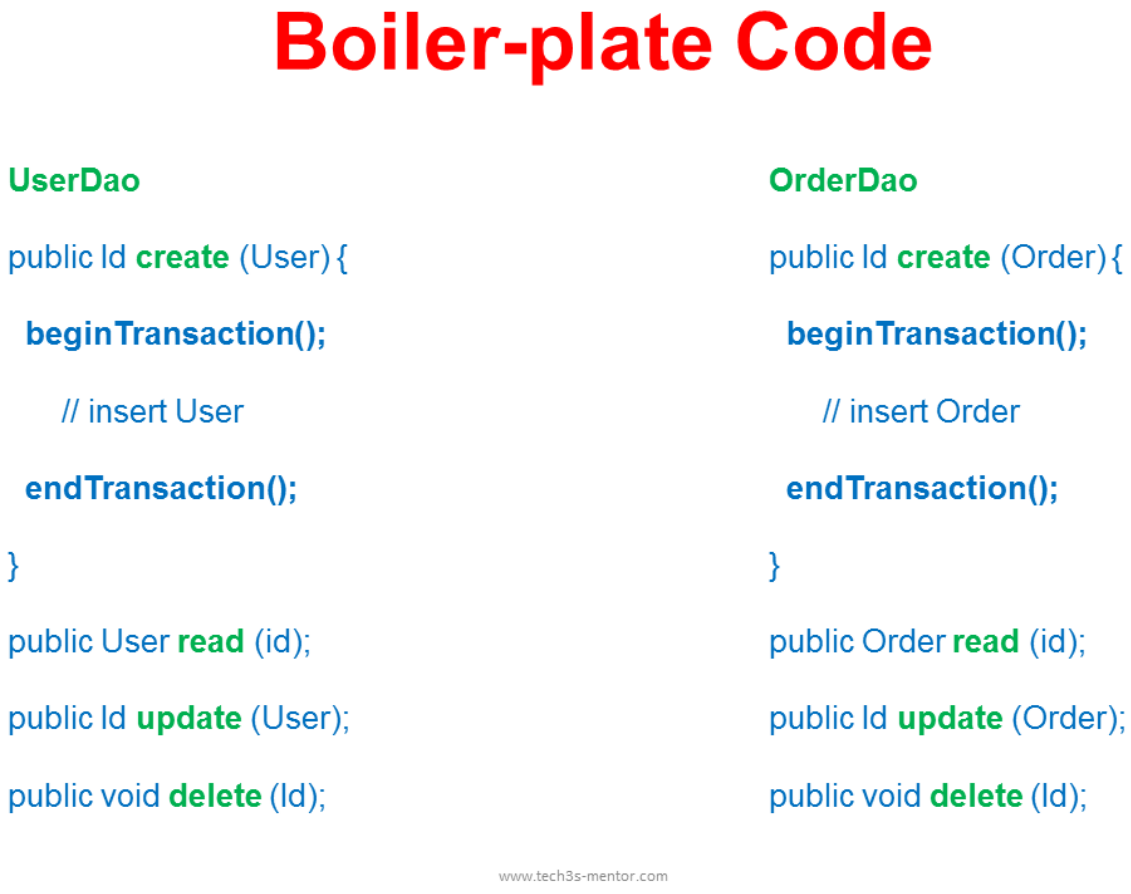
Trên đây chúng ta đã tìm hiểu một số Kiến thức chuyên môn để phát triển phần mềm chuyên nghiệp. Tiếp theo chúng ta sẽ tìm hiểu một số cách để tăng chất lượng phần mềm.

4) Chất lượng mã (Code Quality)

Để 1 dự án phần mềm chạy tốt ngoài việc thiết kế tốt thì việc lập trình ra những đoạn code tốt đóng vai trò rất quan trọng. Như vậy, code tốt được đánh giá dựa trên những tiêu chí gì?

Thực tế là mỗi SCRUM Team đều có các qui định, các chuẩn để các SD theo và họ có các cách đánh giá chất lượng code khác nhau. Tựu trung thì cũng có một số nguyên tắc chung sau đây để đánh giá chất lượng code:

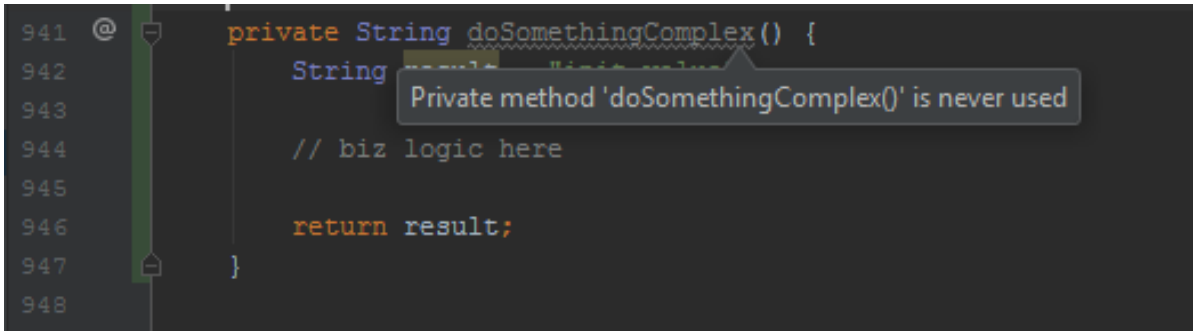
- **Boiler-plate Code**: Điều này chỉ đến việc các đoạn code bị trùng nhau ở những chỗ khác nhau trong khi làm tốt hơn là phải sử dụng lại đoạn code được viết đầu tiên.
  Ví dụ:



Ta thấy 4 hàm create/read/update/delete của 2 đối tượng (User, Order) rất giống nhau mà phải tạo 2 files và định nghĩa đến 8 hàm. Trong trường hợp này ta dùng "Generics" thì chỉ cần 1 file (GenericDao) định nghĩa ra 4 hàm cho tất cả các loại đối tượng như User, Product, Order, Shipment…

- **Redundant Code**: Điều này nói đến việc bạn viết ra các đoạn code nhưng sau đó vì lý do gì đó không xài chúng nhưng cũng không xóa chúng đi.
  Một trường hợp khác nữa là bạn viết ra các đoạn code được xài trong phiên bản X, sau đó qua phiên bản X+1 chỉnh sửa một số chỗ dẫn đến các đoạn code này không còn được xài nữa mà bạn không nhớ hoặc không hay biết.

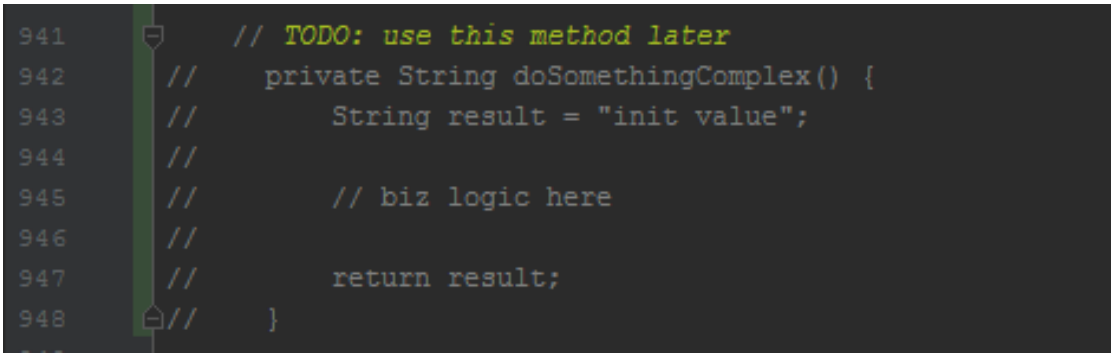
  Ví dụ: Trong **môi trường phát triển tích hợp (Integrated Development Environment – IDE)**, bạn sẽ thấy nó hiển thị tên của phương thức không được sử dụng màu xám đen (‘doSomethingComplex()’), khi bạn rờ chuột vào, nó sẽ hiển thị thông báo.



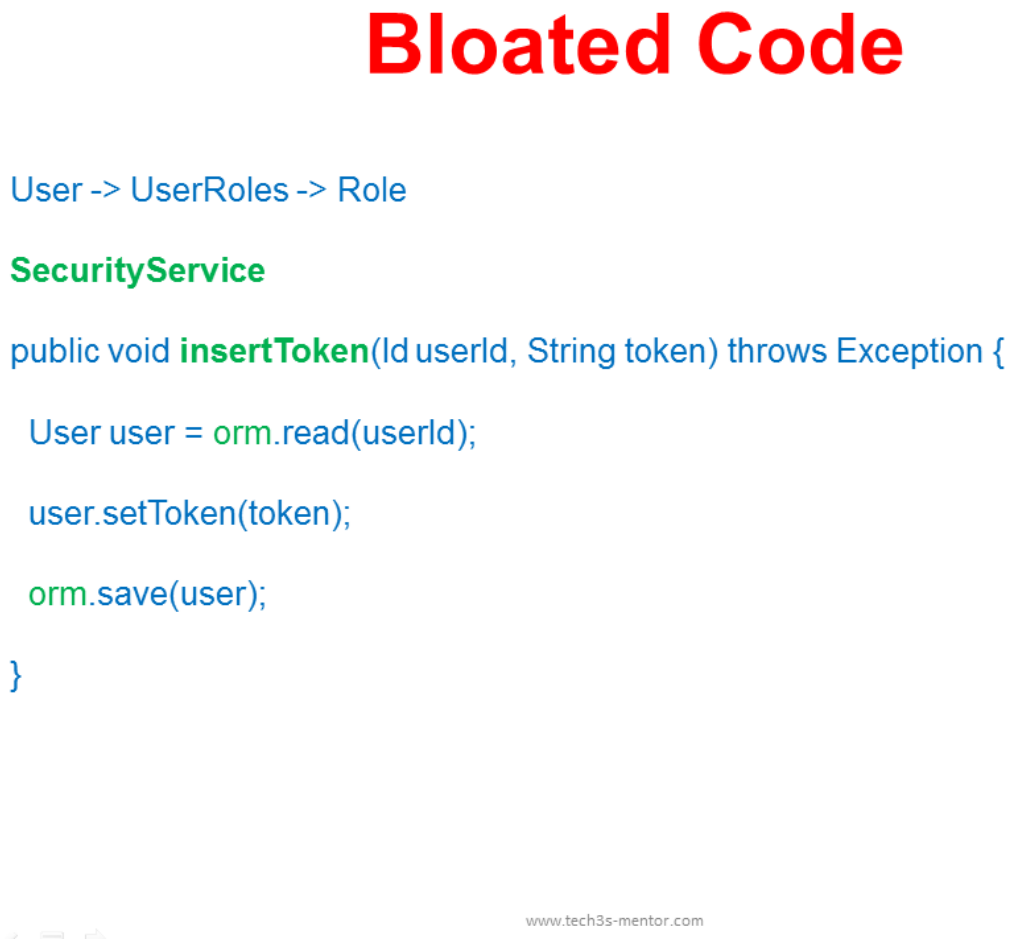
  Giải pháp cho vấn đề này:

  - Xóa phương thức này. Lưu ý: phương thức này có từ khóa ‘private’ mà không được xài thì mới xóa nó, nếu từ khóa là ‘public’ or ‘protected’ hoặc không có (default là ‘package’ đối với ngôn ngữ Java ví dụ) thì không được xóa nó vì nó có thể được xài từ các đối tượng khác bên ngoài sau này.

  - Tạm thời ẩn nó đi và ghi chú lại cho bạn và người khác biết nó sẽ hữu dụng sau này.

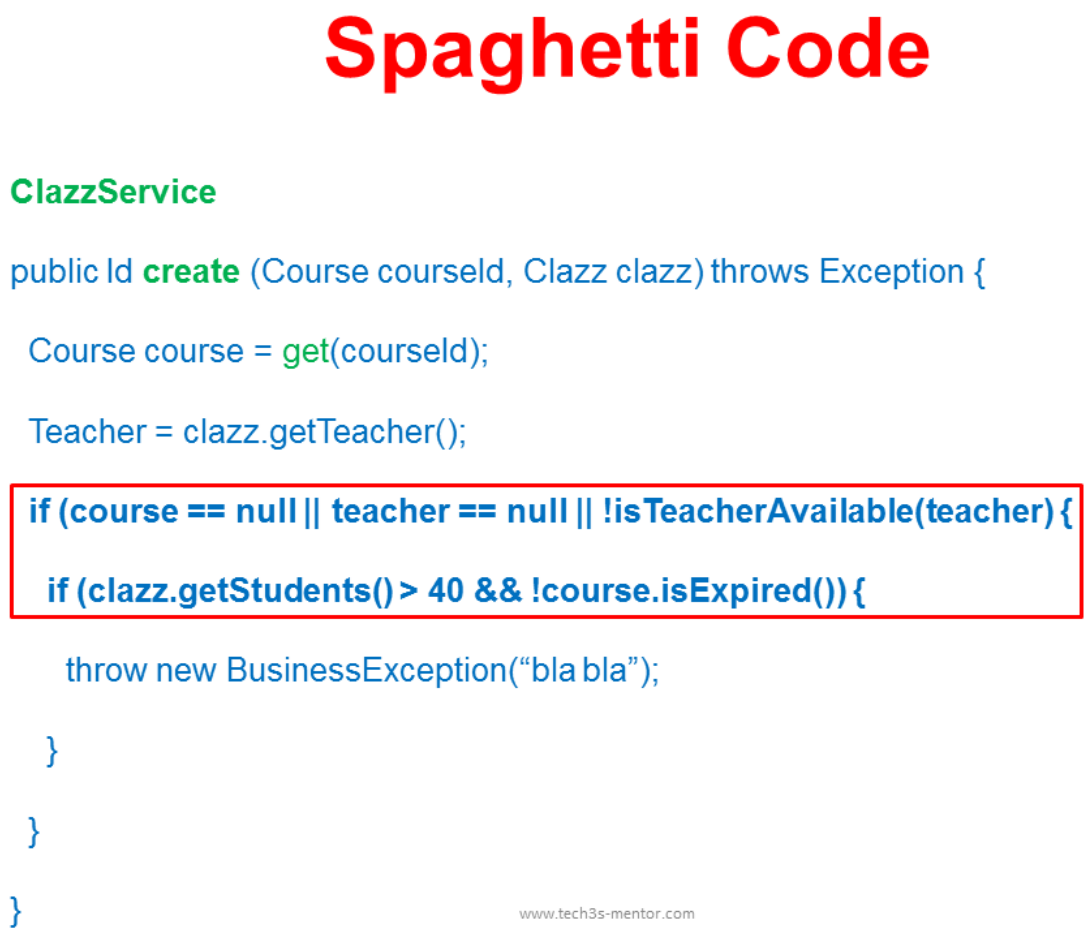


- **Bloated Code**: Cái này nói nôm na theo tiếng Việt là "Đem dao mổ trâu đi giết gà"; ngụ ý cho việc dùng 1 giải pháp rất to lớn để giải quyết 1 vấn đề rất nhỏ.
  Ví dụ:



Ở đây xài công nghệ **Object Relational Mapping (ORM)** để đọc nguyên 1 User object (có thể có rất nhiều thuộc tính) từ CSDL lên, gán giá trị cho 1 thuộc tính là 'token' rồi lại lưu nguyên 1 User object xuống CSDL.
Giải pháp đơn giản và nhanh hơn nhiều là dùng 1 câu SQL để cập nhật giá trị "token" cho đối tượng User trong CSDL là ok.

- **Spaghetti Code**: Spaghetti là tên của món mì Ý với các sợi mì quấn với nhau rất rối, cho nên Spaghetti Code nói đến việc viết code rất rối rắm như mì Ý. Ví dụ:



Đoạn code này có 2 câu if lồng nhau và các điều kiện gộp vào nhau rất rối, khó phát hiện lỗi. Chúng ta nên tách các điều kiện ra, dù viết dài hơn một chút nhưng sau này bảo trì và phát hiện lỗi dễ dàng hơn nhiều so với viết ngắn gọn kiểu rối rắm này.

Một phần mềm khi đưa đến tay người sử dụng thì ngoài tính năng chạy ổn, đáp ứng tốt nhu cầu người dùng thì **hiệu năng (Performance)** của phần mềm cũng là vấn đề đáng lưu tâm. Dễ thấy giữa 2 phần mềm tính năng tương đương nhau, một phần mềm tính năng được đánh giá tốt hơn tí nhưng hiệu năng hơi kém thì sẽ gây không thiện cảm cho người dùng.

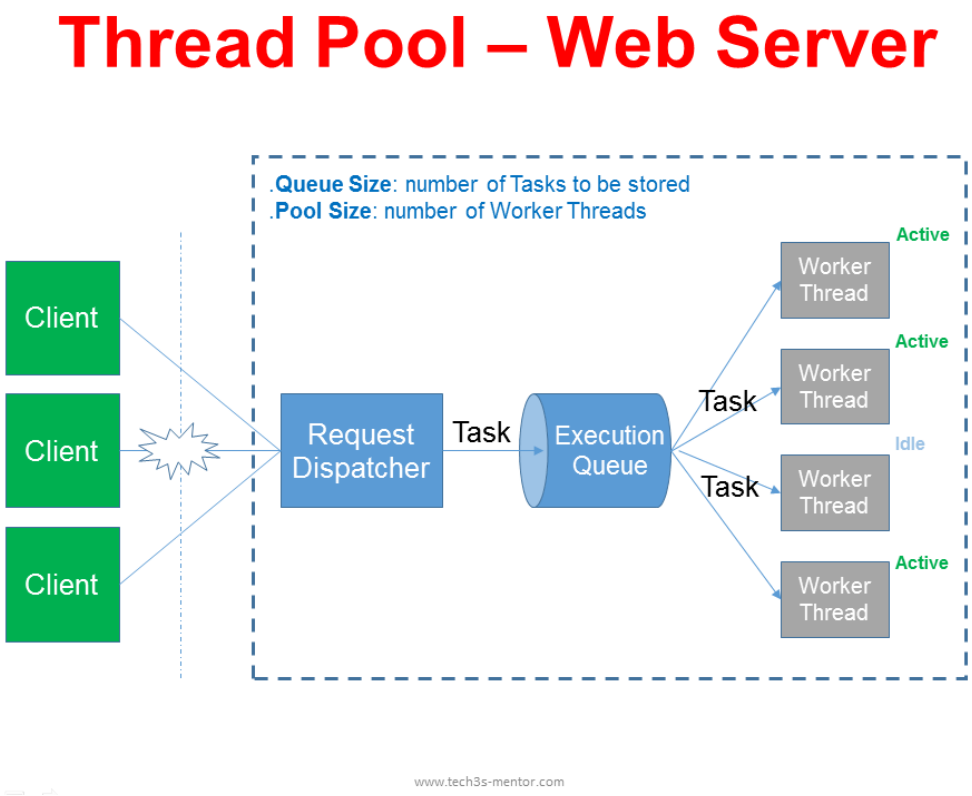
5) Vấn đề hiệu năng (Performance Problem)

Như đã nói ở trên, phát triển phần mềm ngoài việc xử lý FR để cung cấp các tính năng mà người dùng mong muốn một cách đúng đắn nhất thì cũng phải xử lý NFR để hệ thống IT hoạt động hiệu quả nhất, đem đến **trải nghiệm người dùng (User Experience – UX)** tốt nhất.

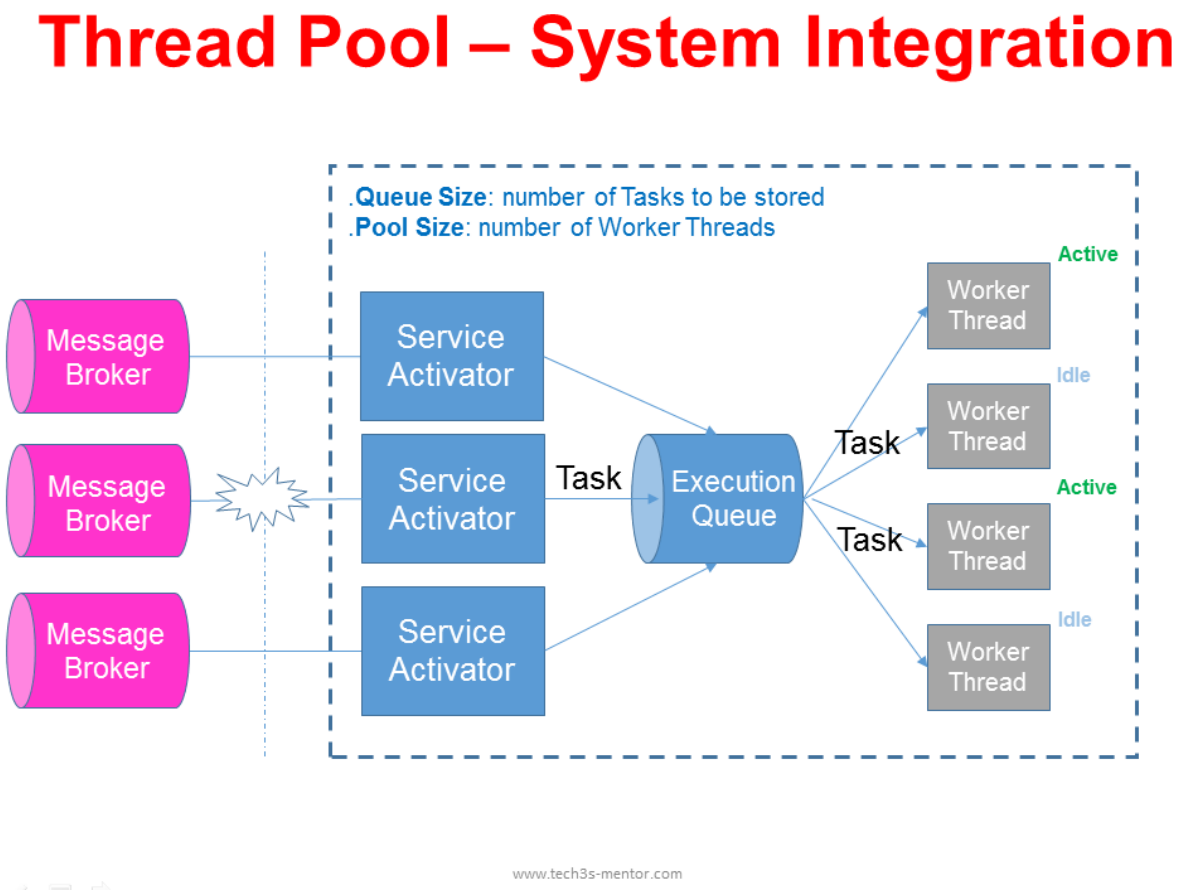
Một trong các NFR khá quan trọng là **hiệu năng (Performance)**. Tính năng này đảm bảo hệ thống IT chạy đúng khả năng của nó mà không bị chậm, ảnh hưởng đến người dùng. Sau đây tôi sẽ chia sẻ một số vấn đề hiệu năng phổ biến.

- **Bloated FE**: Phần FE quá cồng kềnh không cần thiết như:
  - Quá nhiều hình ảnh và/hoặc hình ảnh có kích thước quá lớn
  - Sử dụng không đúng **bộ nhớ đệm của trình duyệt (Browser Cache)**
  - Lạm dụng quá nhiều Javascript hoặc AJAX
- **Thread Pool of web server**: Mỗi phần mềm khi chạy trong HĐH thường có 1 **tiến trình (Process)**, có thể có nhiều **tiểu tiến trình (Thread) – TTT** giúp xử lý các tác vụ song song, trong đó tồn tại 1 TTT chính (Main Thread).

  Trong trường hợp web server, khi có nhiều người dùng gửi yêu cầu đến web server, nếu nó chỉ có 1 TTT chính thì nó không thể phục vụ nhiều người dùng cùng lúc được thì sẽ làm web server chậm, nên nó phải sử dụng nhiều TTT để xử lý vấn đề này. Tuy nhiên mỗi lần có yêu cầu đến mà nó tạo 1 TTT thì rất tốn tài nguyên và làm chậm hệ thống, việc tái sử dụng TTT là cách làm hợp lý hơn. Khái niệm **Thread Pool** xử lý vấn đề này.

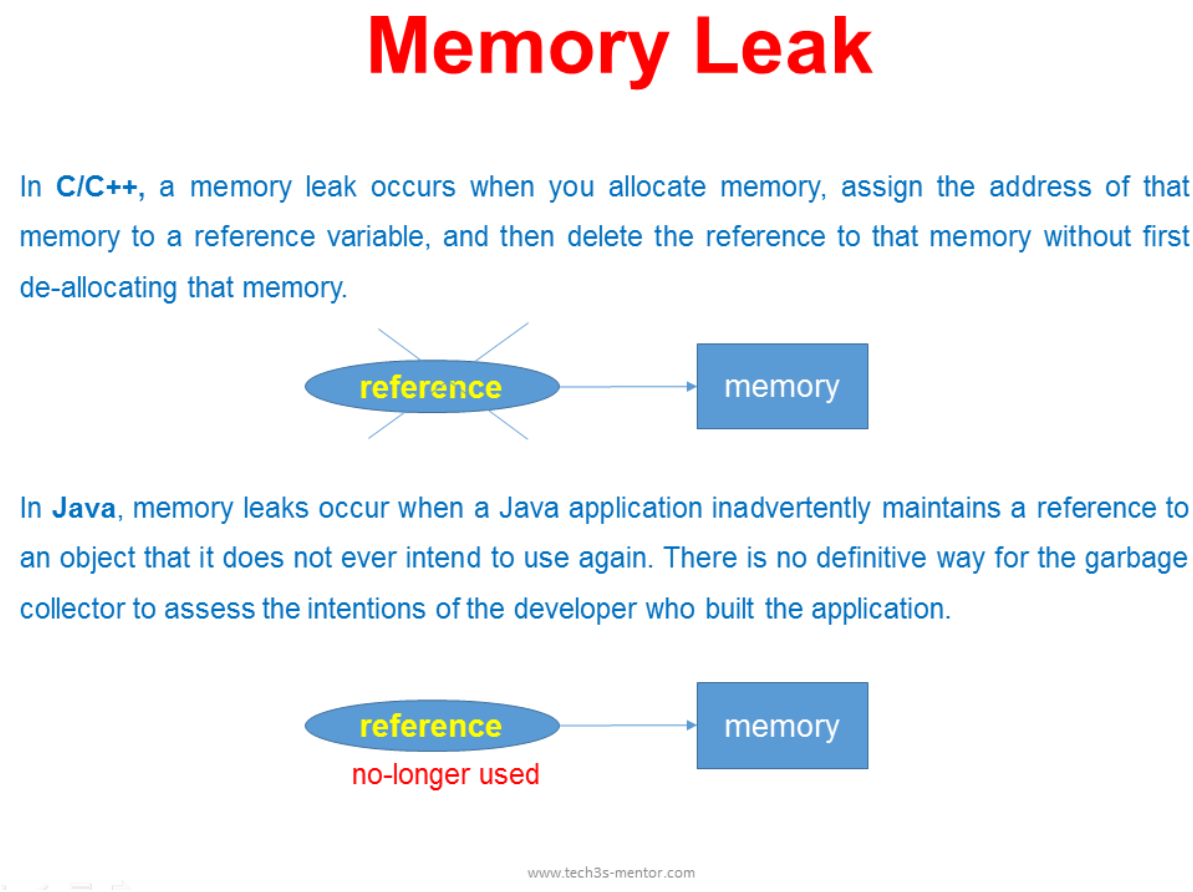


- **Thread Pool in system integration**: Ngoài trường hợp web server thì Thread Pool cũng được xài trong các bài toán về tích hợp hệ thống sử dụng khái niệm **hàng đợi tin tức (Messaging Queue).** Bạn có thể hình dung là hệ thống bạn đang phát triển cần nhận tương tác/dữ liệu từ 1 hệ thống khác. Tuy nhiên, hệ thống của bạn không phải lúc nào cũng rãnh để xử lý các yêu cầu của hệ thống đó một cách tức thì, hay còn gọi là **đồng bộ (Synchronous)**, vì thế bạn có thể xài các message queues để lưu yêu cầu rồi sau đó định thời xử lý các yêu cầu đó một cách **bất đồng bộ (Asynchronous)**. Ví dụ về tích hợp hệ thống sử dụng Thread Pool:



Dựa trên lưu lượng thực tế mà cấu hình 2 thông số queue_size và pool_size phù hợp nhất để đạt được hiệu năng tối ưu.

- **Memory Leak**: Thất thoát bộ nhớ



Vấn đề thất thoát bộ nhớ là vấn đề cực kỳ phổ biến khi lập trình, cho nên bạn phải lưu ý vấn đề này, và tùy theo NNLT bạn sử dụng mà có các chiến lược xử lý vấn đề này một cách tốt đẹp nhất.

- **Structured Query Language – SQL**: Vấn đề này liên quan đến việc viết các câu lệnh SQL không tốt, dẫn đến chạy rất chậm làm giảm hiệu năng của hệ thống.

- **N+1 Select Problem**: Nếu bạn dùng giải pháp **ORM (Objecct-Relational Mapping)** cho phần tương tác với CSDL quan hệ thì lưu ý vấn đề "N+1 Select" khi bạn lấy danh sách dữ liệu và các tập con của các phần tử trong danh sách đó.

6) Xung đột mã (Code Conflict)

Như đã nói ở trên, các SD sẽ làm việc trên cùng 1 thư mục gốc của dự án phần mềm, còn được gọi là **bộ mã nguồn (Souce Code Repository)**.

Khi 1 Sprint đang chạy, các SDs được giao các USs khác nhau. Việc các SDs cùng sửa 1 hay nhiều files trên bộ mã nguồn xảy ra như chuyện hằng ngày ở huyện. Nếu may mắn các phần sửa của họ không đụng nhau thì VCS sẽ gộp những phần sửa lại một cách tự động, còn không may những phần sửa đụng nhau thì các SDs đưa phần code của mình vô bộ mã nguồn sau phải có trách nhiệm xử lý xung đột. Trong quá trình xử lý có những đoạn code nào do SDs trước viết mà không hiểu rõ thì nên liên hệ với họ để xử lý cho chính xác.

Một tình huống khác có khả năng gây ra xung đột nữa.
Đó là khi chúng ta đã đưa phiên bản X cho người dùng xài, tiếp đó SCRUM Team tiếp tục làm phiên bản X+1.
Trong quá trình xài phiên bản X, người dùng báo một số lỗi gặp phải. Thế là các SDs sẽ thay nhau xử lý các lỗi này trên phiên bản X. Sau khi sửa lỗi xong và người dùng đã xài ổn định phiên bản X thì những phần sửa này sẽ được gộp vô phiên bản X+1 để làm tiếp.
Lúc gộp này sẽ có thể gây ra xung đột giữa phần sửa trên phiên bản X và phần SDs đang phát triển trên phiên bản X+1.
Trường hợp này thì các SDs có liên quan đến xung đột phải ngồi lại với nhau xử lý.

Việc xung đột mã xảy ra rất thường xuyên trong các SCRUM Team, do vậy kỹ năng xử lý xung đột này khá là quan trọng giúp cho việc phát triển nhanh hơn và hạn chế lỗi tốt hơn.

Ngoài các Kỹ năng chuyên môn ở trên thì một số Kỹ năng mềm dưới đây cũng rất quan trọng giúp bạn làm việc hiệu quả trong công việc của bạn cũng như thúc đẩy SCRUM Team phát triển ngày càng mạnh mẽ.

III. Kỹ năng mềm

1) *Tiếng Anh làm việc (Professional English)*

Trong thế giới IT, tiếng Anh là cầu nối chúng ta đến với kho kiến thức IT khổng lồ và phát triển không ngừng của nhân loại cũng như phá bỏ khoảng cách giữa những con người đến từ những nền văn hóa, ngôn ngữ khác nhau, hòa vào chung một môi trường làm việc chuyên nghiệp quốc tế. Do đó, rèn luyện tiếng Anh phục vụ cho công việc IT là điều bắt buộc đối với bất cứ dân IT nào. Nhưng luyện tiếng Anh thế nào là hiệu quả?

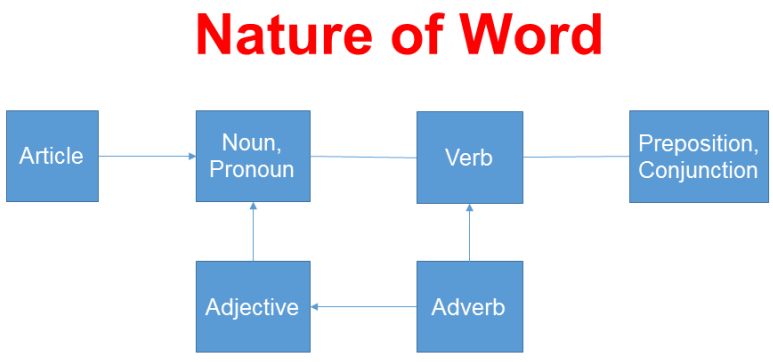
Câu hỏi trên là nỗi trăn trở của biết bao nhiêu người Việt Nam nói chung, dân IT Việt Nam nói riêng, trong đó tôi cũng không là ngoại lệ lúc tôi mới tốt nghiệp đại học bước vào môi trường làm việc ngoài đời. Và thực tế là tôi đã rất vất vả, nhiều lúc rất nản, trong việc luyện tiếng Anh để phục vụ công việc IT của tôi suốt nhiều năm.

Giờ đây tôi đã tổng hợp được một số kiến thức/kinh nghiệm luyện tiếng Anh phục vụ cho công việc IT để chia sẻ đến các bạn IT với hi vọng sẽ giúp các bạn nâng cao hơn kỹ năng tiếng Anh của các bạn và tiết kiệm thời gian cho các bạn.

Kỹ năng tiếng Anh mà bạn phải đầu tư nhiều vào là **Reading (Đọc hiểu) & Writing (Viết)**.

Bạn cần nhớ và **thực hành đều đặn** một số qui tắc quan trọng dưới đây:

- Các cụm từ Tiếng Việt viết từ trái qua phải và hiểu theo theo vậy. Nhưng tiếng Anh viết từ trái qua phải nhưng hiểu từ phải qua trái. Ví dụ: ***Nhiệm vụ quan trọng cực kỳ*** vs ***The extremely important task;*** **Management Information >< Information Management.**
- Mỗi từ tiếng Anh sẽ có bản chất là một trong các loại sau:



1. Article → Noun, Pronoun: **a** database, **the** framework, users, **the** other(s)
2. Adjective → Noun, Pronoun: **big** data, **small** ones
3. Adverb → Adjective: **extremely** critical, **really** needed
4. Adverb, Adjective → Noun: **really needed** requirement
5. Adverb → Verb: **perfectly** match, work **charmingly**

Hiểu được bản chất của từ và sự bổ nghĩa cho nhau của các từ sẽ giúp bạn đọc hiểu đúng (cũng như viết đúng), ví dụ: Make things different. → ở đây chữ "different" bổ nghĩa cho chữ "things" nên câu này hiểu là "Hãy làm những điều khác biệt" vs Make things differently. → ở đây chữ "differently" bổ nghĩa cho chữ "Make" nên câu này hiểu là "Hãy làm những điều bằng cách khác nhau".

- Cùng là tính từ nhưng **-ing** mang nghĩa chủ động, **-ed** mang nghĩa thụ động. Ví dụ: She is bor**ing** (Cô ấy thì chán lắm, ý là cô ấy không có gì thú vị) vs She is bor**ed** (Cô ấy trông buồn chán, chắc cô ấy đang có gì đó không vui).
- Các danh từ ghép trong tiếng Anh rất đa dạng, hiểu được các cách ghép này là **chìa khóa** giúp bạn hiểu được các câu dài và phức tạp. Tựu trung có 1 số các công thức của danh từ ghép sau đây:

NoW – Compound Noun

1. **Adj + N** : poor design, clean code, high cohesion, loose coupling
2. **N + Adj + N** : computer-based test, tree like structure, time-consuming task
3. **Adv + Adj + N** : recently modified set, extremely critical issue, quite nice feature
4. **N + N** : project management, document title, information system
5. **Adj + N + N** : statistical analysis method, approximate calculation algorithm

Example:

We usually analyze recently modified linked list like data structure highly used in management information system in designing information management system.

We usually analyze recently modified linked list like **data structure** highly used in management information system in designing information management system.

www.tech3s-mentor.com

➔ Bạn hãy dịch thử cái ví dụ trên trước khi xem tiếp phần dịch bên dưới nhé…

.

.

.

(Chúng tôi thường phân tích cấu trúc dữ liệu có dạng như danh sách liên kết được thay đổi gần đây, mà cấu trúc này được sử dụng nhiều trong hệ thống thông tin quản lý, trong việc thiết kế hệ thống quản lý thông tin).

- Mỗi từ tiếng Anh có 1 bản chất duy nhất, nhưng khi đứng trong 1 câu thì nó có thể có nhiều chức năng. Hiểu được chức năng từ giúp chúng ta sử dụng đúng từ khi viết.

Function of Word

A word in sentences can have different functions.

Example: A noun can be a Subject also an Object.

- **Water melon** is very delicious. → Subject
- I like to eat **water melon** a lot. → Object

- Như tiếng Việt, Tiếng Anh có 2 loại câu: câu đơn & câu ghép. Câu đơn có 9 loại.

Simple Sentence

1. Subject + Verb + **Subject Complement**.
2. Subject + Verb + **DO**.
3. Subject + Verb + **DO** + **Bare Infinitive**.
4. Subject + Verb + **DO** + **To Infinitive**.
5. Subject + Verb + **To Infinitive**.
6. Subject + Verb + **Gerund**.
7. Subject + Verb + **DO + to IO**.
8. Subject + Verb + **IO + DO**.
9. It's **Adj** for **Object to Infinitive**.

- Câu ghép cũng có 2 loại: câu ghép đơn & câu phức hợp. Câu ghép đơn có 2 mệnh đề độc lập nhau và kết nối với nhau bởi 3 từ thông dụng: **and**, **but**, **so**.

Compound Sentence

1. Developers implement features, **and** QCs prepare test cases.
2. There were a lot of issues during the sprint, **but** we have just finished the sprint well.
3. There are some misunderstandings among teams, **so** we need a meeting to clarify.

- Câu phức hợp có 2 mệnh đề chính và phụ. Mệnh đề phụ có 3 loại:
  - **Mệnh đề tính từ (Adjective Clause)**

Complex Sentence – Adj Clause

1. Subject (people): who/ that/ present participle

- I have just met a PM **who** has great management skill. (*defined*)
- James Gosling, **who** is the father of Java, has left Oracle. (*non-defined*)
- People **that** have design ideas like yours are really rare.
- User **having** read privilege can access that folder.

2. Subject (animals/things): which/ that/ past participle/ -

- It is the subject **which/that** interests me.
- The directory **used** to store source code is nearly full.
- Flan, **a** kind of cake, is tasty.
- Note: I have read several articles, **most of which** were useful.

3. Direct Object (people/animals/things): whom/ which/ that/ -

- The man **[whom]** you met yesterday is the BA of our team.
- The Java book **[which/that]** I bought last week was very expensive.
- The monitoring tool we have used so far is extremely helpful. (- : *omitted*)

4. Indirect Object (people/animals/things): preposition + whom/which

- That is the client representative **to whom** you have talked.
- It is an interesting project **on which** I will work.
- She has suggested a topic **[which]** we have been discussing **about**.

5. Noun Complement: whose

The serializable object is the one **whose** properties are all serializable.

6. Adverbial Complement: when, where

- The season **when** we organize technical events is the summer.
- That is the directory **where** we store all configuration and log files.

- **Mệnh đề danh từ (Noun Clause)**

Complex Sentence – Noun Clause

- I was happy **that** I had won the lottery.
- I am happy **that** I have won the lottery.
- I was excited **to go** to the cinema last night.
- I don't know **if** her office is on this floor.
- Please tell me **where** we should push our commits to.
- I don't know **what** I have to do to fix that issue.

- Mệnh đề trạng từ (Adverb Clause)

Complex Sentence – Adv Clause

1. Time Clause:

- **Before** coming Canada, I didn't know French.
- I started learning French **when** I came to Canada in 2006.
- I have been learning French **since** 2006.
- I took 2 French courses 2 years **ago**.
- I have been learning French **for** 3 years.

2. Condition Clause:

- **If** I **had known**, I **wouldn't have waited** for her.
- **If** I were a millionaire, I **would buy** that de lux car.
- **If** I **can register** for that course, we **will study** together.
- I **will go** out **on the condition that** the weather **is** good.

3. Cause Clause:

- The code is **so** messy **that** we can not maintain.
- The code structure is **too** spaghetti **to** refactor.
- **Thanks to** QC and development teams, we can release on time.
- We can not connect to the server **because of** down time.

4. Purpose Clause:

- I usually practice designing software architecture **so that** I can become a TA.
- We should refactor the source code **in order to/to** maintain better.

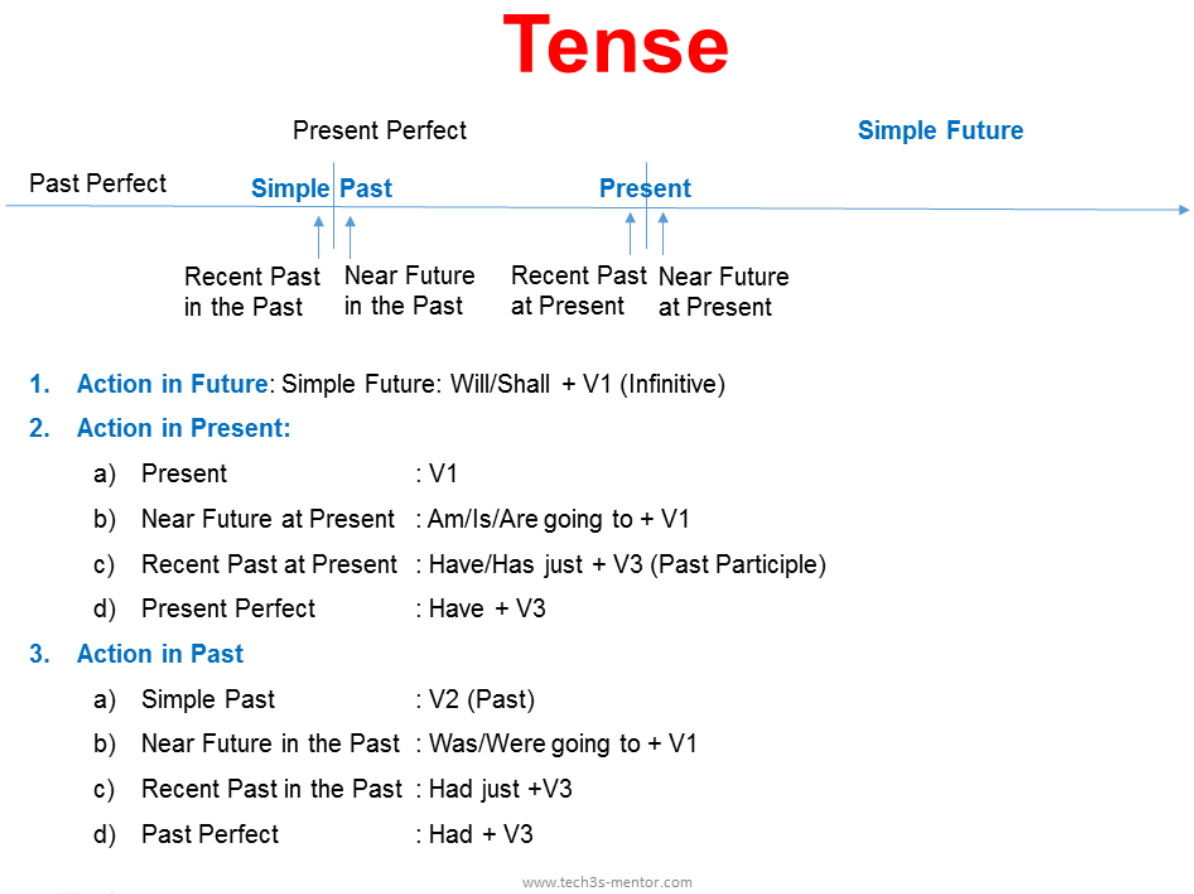
5. Contrast Clause:

- **Although** we have tried the hot fix, the system is still buggy.
- Memcached does not support set operation, let's use Redis **instead**.
- We can deploy the code change **without** having to restart the server.

www.tech3s-mentor.com

- Tiếng Việt chỉ có 3 thì và cách thành lập cũng rất đơn giản: thêm từ "đã" chỉ quá khứ, bình thường là hiện tại, còn tương lai thêm từ "sẽ". Nói đến thì tiếng Anh (English tense) thì chắc ai từng học cũng phải mệt não vì có quá nhiều thì và cách dùng cũng quá lộn xộn, đôi khi cách dùng trùng lắp lên nhau nên có tình huống chả biết dùng thì nào là đúng.

Do đó tôi đã tổng hợp các thì được xài phổ biến một cách đơn giản hơn qua lược đồ sau:



- Hành động xảy ra trong tương lai ta xài thì tương lai đơn cho đơn giản, các thì tương lai khác rất ít khi được xài.
- Hành động xảy ra ở hiện tại thì ta lấy thì **hiện tại đơn (Simple Present)** làm mốc thời gian cho hành động. Nhích tới 1 tí ta có **thì tương lai gần ở hiện tại (Near Future at Present)** vs Lùi về 1 tí ta có **thì quá khứ gần ở hiện tại (Recent Past at Present)**. Và cuối cùng là **thì hiện tại hoàn thành (Present Perfect)** diễn tả hành động xảy ra trong quá khứ rồi kéo dài đến hiện tại hoặc/và không rõ thời gian. Ví dụ: **I have worked on the project for 3 years.** → Tôi đã làm dự án đó được 3 năm. (Hiện giờ tôi vẫn còn làm trên dự án đó); **She has been the good girl this year, so she deserves a Chrismast gift.** → Cô bé đã là cô gái tốt suốt năm nay nên cô ấy xứng đáng một món quà Nô en. (giờ cô bé vẫn tốt nhưng không biết cô bé bắt đầu tốt khi nào, có thể trước đó không tốt).
- Hành động xảy ra trong quá khứ thì ta lấy **thì quá khứ đơn (Simple Past)** làm mốc thời gian cho hành động (hoặc 1 trạng từ chỉ thời gian trong quá khứ cụ thể). Tương tự như mốc hiện tại ở trên, ta cũng có 3 thì tương ứng: tương lai gần trong quá khứ vs quá khứ gần trong quá khứ và cuối cùng là quá khứ hoàn thành.

- Văn phong khác nhau giữa tiếng Anh & Việt gây hiểu nhầm. Ví dụ:
  - Won't you do that task? → Yes, I do (Tôi làm) vs No, I don't (Tôi không làm).
  - Bạn không làm nhiệm vụ đó hả? → Ừ (Tôi không làm) vs Không (Tôi làm).

  → **Cách trả lời của tiếng Anh thì không quan tâm câu hỏi là phủ định hay khẳng định** trong khi tiếng Việt thì cần quan tâm câu hỏi.

- Đặc biệt chú ý đến các từ/cụm từ nhìn gần giống nhau nhưng ý nghĩa khác nhau dẫn đến hiểu sai ý người nói, đôi khi gây hậu quả nghiêm trọng.
  - **Try to open** the door vs **Try opening** the door: Cố gắng mở cánh cửa (chắc do khó mở hay bị khóa) vs Thử mở cánh cửa (chắc đang tìm gì đó).
  - **Stop to smoke** vs **Stop smoking**: Dừng lại để hút thuốc vs Dừng việc hút thuốc (bỏ hút thuốc vĩnh viễn).
  - **Forget to shut down** the server vs **Forget shutting down** the server: Quên tắt server (server chưa bị tắt → server vẫn còn chạy) vs Quên việc tắt server (server đã bị tắt nhưng người tắt quên béng việc tắt này → server hết chạy).

Kỹ năng **nghe hiểu tiếng Anh (English listening)** quan trọng khi bạn làm việc trực tiếp với người nước ngoài hoặc tham dự các hội thảo bằng tiếng Anh. Chúng ta gặp rất nhiều khó khăn để nghe hiểu người nước ngoài nói trong môi trường làm việc thực tế vì các lý do sau:

- Những câu/từ tiếng Anh chúng ta học ở các trường/trung tâm ở Việt Nam hầu hết là các từ hàn lâm (dành cho học thuật) trong khi bên ngoài người ta dùng nhiều thành ngữ (idioms, expressions), từ lóng (slang words).
- Các bài nghe chúng ta học ở các trường/trung tâm ở Việt Nam hầu hết là những câu chuyện được xây dựng chứ không phải những câu chuyện thực tế và các nhân vật trong các câu truyện thì ĐỌC với tốc độ vừa phải chứ không phải NÓI với tốc độ bên ngoài thực tế (khá là nhanh và nuốt chữ nhiều).
- Chúng ta thiếu từ vựng nên không biết những từ họ nói phát âm như thế nào thì làm sao nhận biết được chúng khi chúng ta nghe.

Do đó, để luyện nghe tiếng Anh cho công việc thì chúng ta phải học tiếng Anh thực tế. Tôi đã từng thử nghiệm chương trình học tiếng Anh thực tế này 1 năm và kết quả tôi đạt được rất tốt, bạn có thể thử xem: https://learnrealenglish.com/. Phương pháp này nhìn chung rất hiệu quả nhưng đòi hỏi bạn phải **rất kiên trì và khổ luyện trong thời gian dài**…

Kỹ năng tiếng Anh cuối cùng và cũng khó nhất đó là **nói tiếng Anh (English speaking)**.
Thú thật là từ lúc học tiếng Anh thời phổ thông, rồi đại học, sau đó ra trường đi làm trong môi trường có người nước ngoài và thời gian đầu khi tôi qua Canada du học thì tôi luôn rất ư là ngại nói tiếng Anh. Tôi nghĩ không chỉ riêng tôi mà hầu hết người Việt Nam nói riêng và những người học tiếng Anh trên thế giới như Hàn Quốc, Trung Quốc, các nước Nam Mỹ… nói chung đều ngại nói tiếng Anh. Tại sao vậy?

Từ kinh nghiệm thực tế tôi đúc kết ra được một số lý do sau:

- Sợ nói sai thì người bản xứ cười. (Thực tế là họ không bao giờ làm thế, ngược lại họ còn chăm chú lắng nghe và hỗ trợ mình để hiểu mình nói gì).
- Tâm lý run (như tham dự phỏng vấn, thảo luận chuyện quan trọng…) nên đầu óc không sản sinh ra được chữ nghĩa gì ráo để nói.
- Không có môi trường để được nói tiếng Anh hàng ngày nên dễ bị cứng miệng.
- Không có đủ từ vựng để diễn đạt ý mình muốn.

Tôi có một số lời khuyên giúp bạn cải thiện kỹ năng nói tiếng Anh của bạn:

- **Hãy nói tiếng Anh với suy nghĩ tiếng Anh trong đầu**, đừng suy nghĩ tiếng Việt sau đó dịch ra tiếng Anh trong đầu rồi mới nói thì không kịp thời gian đâu; thứ hai là suy nghĩ bằng tiếng Anh giúp "cảm" được ngôn ngữ/văn hóa, gọi là sắc thái của ngôn ngữ (nuance). Trải nghiệm thực tế của tôi khi học tiếng Pháp ở Canada thì lúc đầu tôi suy nghĩ tiếng Anh trong đầu, sau đó dịch qua tiếng Pháp làm cho khả năng nói của tôi rất chậm, sau khi nghe lời khuyên của 1 người bạn thì tôi đã suy nghĩ bằng tiếng Pháp khi nói, kết quả là khả năng nói tiếng Pháp của tôi cải thiện rất nhiều so với lúc đầu.
- **Hãy nói một cách tự nhiên, không cần hay** (như giọng tôi không thể nói tiếng Anh/Pháp hay hoặc bay bổng - miễn đối phương hiểu đúng ý), **không sợ sai** bằng cách thực hành nói hàng ngày trước gương, đừng quan tâm quá nhiều đến ngữ pháp vì văn nói với văn viết khác xa nhau lắm, quan tâm đến câu cú này nọ sẽ làm chậm phản xạ của bạn khi nói.
- **Hãy tìm mọi cách nói tiếng Anh càng nhiều càng tốt với người bản xứ**, hạn chế nói tiếng Anh với người không phải bản xứ, và **bắt chước** thật nhiều phát âm, từ vựng cũng như cách nói của người bản xứ để sau này nói với người bản xứ khác thì khả năng họ hiểu bạn sẽ rất cao.
- **Tích lũy từ vựng thực tế mỗi ngày** qua phim ảnh (phim đời thường, đừng xem phim nghệ thuật hay hành động khó hiểu hoặc nhanh quá thì khó nắm bắt) và ráng kiên trì/khổ luyện chương trình ở phần nghe bên trên.

2) Suy nghĩ luận lý (Logical Thinking)

Trong cuộc sống chúng ta, mọi sự việc xảy ra đều tuân theo qui luật nào đó. Cho nên khi chúng ta làm việc gì cũng cần suy nghĩ đến tính hợp lý. Nói một cách khác là chúng ta suy nghĩ luận lý một vấn đề nào đó dựa trên các **dữ kiện (Fact)**, **kiến thức/kinh nghiệm (Knowledge/Experience)** của chúng ta để đưa ra được một **kết luận (Conclusion)** hay một **giải pháp (Solution)** nào đó nhằm giải quyết **vấn đề (Problem)** chúng ta đang gặp phải.

Định nghĩa một cách khoa học thì suy nghĩ luận lý là một quá trình tâm lý có tính học hỏi (Learned Mental Process) trong đó con người **suy luận (Reasoning)** liên tục để đưa đến một **kết luận (Conclusion)** gì đó.

Ứng dụng của suy nghĩ luận lý có thể kể đến như sau:

- Phân tích nghiệp vụ (Business Analysis)
- Giải quyết vấn đề (Problem Solving)
- Ra quyết định (Decision Making)

Phương pháp suy nghĩ luận lý thường dùng nhất là: **Câu điều kiện (If Statement)**.
Ví dụ: Giả sử chúng ta biết được là nếu chúng ta xóa file "xyz.sys" thì hệ thống IT sẽ ngưng hoạt động. Khi thực tế xảy ra là hệ thống IT tự nhiên ngưng hoạt động, chúng ta có thể suy luận ra là có ai đã xóa cái file "xyz.sys".

Một ví dụ khác thể hiện tính bắt cầu trong suy luận:
.Nếu tôi đi làm hôm nay thì tôi có thể tham gia cuộc thi viết game.
.Nếu tôi tham gia cuộc thi viết game thì tôi có thể đoạt giải.
→ Nếu tôi đi làm hôm nay thì tôi có thể đoạt giải.

Suy nghĩ luận lý là 1 quá trình 5 bước:

- Đặt ra mục tiêu
- Lên kế hoạch làm sao để đạt mục tiêu đó
- Thu thập thông tin
- Suy luận liên tục dựa trên thông tin đã có, kiến thức/kinh nghiệm của mình
- Kết luận và kiểm tra

Suy nghĩ luận lý giúp SP nắm bắt yêu cầu phần mềm, nghiệp vụ tốt và hiện thực ít lỗi.

3) Giải quyết vấn đề (Problem Solving)

Mỗi ngày chúng ta thức dậy cho đến lúc chúng ta đi ngủ, chúng ta phải giải quyết rất nhiều vấn đề của cá nhân, công việc, xã hội… từ đơn giản đến phức tạp, từ dễ đến khó... Vì vậy vấn đề luôn tồn tại xung quanh chúng ta hàng ngày để chúng ta giải quyết.
Tuy nhiên cách giải quyết vấn đề như thế nào tốt nhất là cả 1 vấn đề, đòi hỏi chúng ta phải luôn rèn luyện kỹ năng giải quyết vấn đề.

Các vấn đề mà một SD gặp trong quá trình làm việc:

- Làm việc với BA để hiểu rõ nghiệp vụ phần mềm
- Hiện thực US mới
- Làm việc với DevOps để **điều tra sự cố (Troubleshoot issues)**
- Làm việc với QC để xác định các lỗi phần mềm
- Sửa lỗi (Fix bugs)
- Hỗ trợ khách hàng (Customer Support)

Phần lớn các SDs Việt Nam không nhận thức được kỹ năng giải quyết vấn đề, nên khi gặp 1 vấn đề (phần lớn là hiện thực USs/sửa lỗi) là họ nhảy vô xử lý liền với giải pháp đầu tiên mà họ nghĩ ra. Vì vậy rất nhiều trường hợp là sửa lỗi chỗ này lại gây ra lỗi chỗ khác bởi vì giải pháp họ đã làm không phải là phù hợp nhất.

Giải quyết vấn đề là 1 quá trình 5 bước:

- Phân tích tất cả dữ kiện liên quan đến vấn đề để tìm ra **nguyên nhân chính (Root Cause)** của vấn đề.
- Tìm tất cả **giải pháp (Solutions)** có thể xử lý nguyên nhân chính này.
- Với mỗi giải pháp tìm được, liệt kê tất cả **ưu/nhược điểm (Pros/Cons)** của nó.
- So sánh & cân nhắc **(Compare & Consider)** các giải pháp dựa trên ưu/nhược điểm của chúng.
- Chọn giải pháp nào PHÙ HỢP NHẤT cho vấn đề dựa trên tình hình hiện tại (lưu ý là không bao giờ có giải pháp hoàn hảo mà chỉ có giải pháp phù hợp nhất thôi).

Kỹ năng giải quyết vấn đề đóng vai trò rất quan trọng với SP vì nó giúp họ làm việc có năng suất cao hơn và giảm thời gian hơn. Kỹ năng này của SP sẽ được cải thiện dần dần theo thời gian dựa trên sự tích lũy kiến thức/kinh nghiệm của họ qua từng dự án.

4) Quản lý thời gian và công việc (Task & Time Management)

Mỗi ngày SD có 8 tiếng làm việc trong SCRUM Team, trong đó 85-88% được xem là **có năng suất (Productivity)**, tức là có đóng góp cho sự phát triển của sản phẩm phần mềm, phần còn lại cho các hoạt động khác như họp hành, trao đổi/thảo luận nhóm, hoạt động cá nhân khác…

Trong thời gian năng suất, SD thường không chỉ làm một việc duy nhất mà sẽ có nhiều việc phải làm đồng thời vì các việc có thể liên quan với nhau và ảnh hưởng đến công việc của các thành viên khác trong SCRUM Team, ví dụ: SD A phải xong Task 1 buổi sáng để SD B làm tiếp Task 2 dựa trên kết quả Task 1, sau đó SD A phải sửa xong con Bug 6 để DevOps triển khai lên môi trường stage để QC D kiểm tra, rồi chiều SD A phải hiện thực US 4 cho DevOps triển khai lên cho BA chạy thử xem thế nào…

Dù cho SD có lập kế hoạch chi tiết cho công việc trong ngày/tuần thế nào thì trong quá trình làm thực tế thì cũng sẽ có những trục trặc từ bản thân SD hoặc từ các thành viên khác trong SCRUM Team. Cho nên kỹ năng quản lý thời gian và công việc của SD cực kỳ quan trọng để đảm bảo tiến độ của bản thân SD cũng như SCRUM Team và dự án. Nếu làm không tốt việc này thì có khả năng SD phải làm thêm **ngoài giờ (Over Time – OT)**, đây là nỗi ám ảnh của rất nhiều SD Việt Nam -_-

Từ kinh nghiệm thực tế, tôi chia sẻ cho bạn vài lời khuyên giúp bạn quản lý thời gian và công việc của bạn tốt hơn:

- Giờ nào việc đó, đừng để thời gian của các việc bị chồng chéo nhau sẽ ảnh hưởng đến chất lượng công việc.
- Việc dễ làm trước cho mã vào bộ mã nguồn trước để hạn chế việc bị xung đột mã với các thành viên khác trong SCRUM Team.
- Khi gặp một vấn đề hóc búa (nhất là những lỗi khó) thì chỉ nên bỏ từ 1-3 tiếng tìm nguyên nhân chính của nó, nếu sau 3 tiếng không ra thì đừng có cố ngồi đó tìm suốt mà hãy đứng lên đi khỏi máy tính một lúc rồi quay lại hoặc làm việc khác dễ hơn rồi quay lại lỗi này.
- Đối với những việc liên quan đến các thành viên khác thì tăng cường giao tiếp với họ để biết thứ tự thực hiện sao cho hợp lý và nhanh nhất, hạn chế bớt việc làm xong bị lỗi rồi phải làm lại từ các bên do thiếu hiểu ý ngay từ đầu.
- Trước khi vô họp phải đọc kỹ nội dung sẽ họp, chuẩn bị kỹ những câu hỏi/thắc mắc sẽ đem ra thảo luận trong buổi họp.

5) Làm việc nhóm (Team Work)

Người ta có câu **"Muốn đi nhanh hãy đi một mình, muốn đi xa hãy đi cùng nhau"**.

Áp dụng câu này vào phát triển phần mềm cũng rất đúng. Xây dựng 1 phần mềm mà chỉ có 1 người làm thì phần mềm đó sẽ được xây dựng nhanh nhưng không thể lớn và giúp con người giải quyết những việc lớn được.

Những phần mềm được xây dựng bởi 1 người hầu hết là các bản **thử nghiệm ý tưởng** (**Proof of Concept – POC**) xem ý tưởng có khả thi trong thực tế hay không. Nếu kết quả khả thi thì sau đó 1 SCRUM Team sẽ nhảy vào phát triển tiếp phần mềm đó cho mạnh lên để đem đến tính năng tốt cho người dùng.

Như vậy, khi đã làm phần mềm thì hiển nhiên SD sẽ làm trong một nhóm. Thế thì kỹ năng làm việc nhóm của mỗi cá nhân sẽ đóng vai trò quan trọng cho sự thành công của nhóm.

Một sự thật đáng buồn nhưng rất đúng, đó là kỹ năng làm việc nhóm của người Việt Nam cực kỳ kém. Từng đã có thực tế 1 người Việt Nam có khả năng hơn hẳn 1 người Nhật hay 1 người Tây, tuy nhiên 3 người Việt Nam làm chung nhóm thì hiệu lại quả thấp hơn nhóm 2 người Nhật hay 2 người Tây. Sở dĩ có nghịch lý thực tế này là vì 3 người Việt Nam có thể đã không cùng nhìn một hướng và phối hợp với nhau không tốt, ngoài ra có thể có yếu tố tính tình không hợp theo kiểu bằng mặt mà không bằng lòng cũng kìm hãm công việc của nhau.

Để một nhóm làm việc tốt thì các thành viên trong nhóm cần có những điều sau:

- Cùng nhìn một hướng
- Đặt lợi ích nhóm trên lợi ích cá nhân
- Tuân thủ kỉ luật nhóm
- Tăng cường giao tiếp trong nhóm để mọi người hiểu nhau hơn
- Giải quyết các mâu thuẫn phát sinh một cách triệt để và ngăn ngừa xảy ra trong tương lai
- Theo lý thuyết lý tưởng của SCRUM thì mọi thành viên trong nhóm đều có khả năng thay thế nhau, nhưng thực tế thì không có nhóm nào đạt được điều này. Cho nên các thành viên cố gắng học hỏi lẫn nhau để sao mỗi vị trí có ít nhất 2 người nhằm đề phòng những trường hợp đột xuất có người bận không làm được thì có người thay thế để không làm chậm tiến độ của nhóm của như tiến độ dự án.
- Lý tưởng hơn nữa là mọi thành viên đều có **Tinh thần làm chủ (Enterpreneurial Spirit)**, tức là dù mình làm công ăn lương hay mình làm sản phẩm cho mình thì luôn nghĩ những giải pháp tối ưu nhất để làm sao cho sản phẩm tốt nhất thì tất cả đều có lợi.

6) Thuyết trình & Giao tiếp (Presentation & Communication)

Trong quá trình làm việc, nhiều lúc SD sẽ phải thuyết trình trình bày giải pháp kỹ thuật của mình cho nhóm để bàn bạc thảo luận nhằm tìm ra giải pháp tối ưu cho vấn đề dự án đang gặp phải hoặc SD đảm nhận vai trò mentor thuyết trình huấn luyện cho các bạn mới vào nhóm.
Khó hơn một mức nữa là SD phải thuyết trình thuyết phục khách hàng bên ngoài sử dụng giải pháp/sản phẩm của công ty mình.

Do đó, SD phải luôn rèn luyện và hoàn thiện dần dần kỹ năng thuyết trình để truyền đạt ý của mình đến người nghe rõ ràng, chính xác.

Một số chia sẻ kinh nghiệm của tôi về thuyết trình giải pháp kỹ thuật:

- Trước khi thuyết trình:
  - Chuẩn bị kỹ slides có cấu trúc chính sau
    - Chủ đề buổi thuyết trình
    - Tên người thuyết trình
    - Các mục sẽ được trình bày (Agenda)
    - Nội dung chi tiết (**nên dùng nhiều hình ảnh hơn chữ**)
    - Cảm ơn & Hỏi đáp
  - Chuẩn bị trước những câu trả lời cho các câu hỏi có thể được hỏi
  - Tập thuyết trình trước gương vài lần cho lưu loát
- Trong khi thuyết trình
  - Cố gắng nói to và rõ
  - Tập xài các động tác tay để diễn đạt thêm trong khi nói
  - Thường xuyên tương tác với người nghe để biết họ có đang theo kịp nội dung mình đang trình bày hay không

Ngoài kỹ năng thuyết trình thì kỹ năng giao tiếp với các thành viên làm các nghề IT khác nhau trong nhóm như BA, QC, DevOps, PM cũng rất quan trọng. Để giao tiếp tốt với họ thì cần hiểu được công việc họ làm như thế nào (xem lại các nghề này trong cuốn cẩm nang "Định hướng nghề nghiệp IT").

Trên đây là những chia sẻ kiến thức/kinh nghiệm thực tế của tôi về các kỹ năng mềm mà SP cần có. Bạn muốn học bài bản hơn thì có thể tìm các khóa học kỹ năng mềm ở các trung tâm.

Phụ Lục

* Danh sách các từ viết tắt

BA	Business Analyst
BE	Backend
CNPM	Công Nghệ Phần Mềm
CSDL	Cơ Sở Dữ Liệu
CTDL	Cấu Trúc Dữ Liệu
FE	Frontend
FR	Functional Requirement
FW	Framework
HĐH	Hệ Điều Hành
MMT	Mạng Máy Tính
MPA	Multiple Page Application
MVT	Máy Vi Tính
NFR	Non-Functional Requirement
NNLT	Ngôn Ngữ Lập Trình
QC	Quality Control
SA	Software Architect
SD	Software Developer
SP	Software Professional
SPA	Single Page Application
UI	User Interface
US	User Story
UT	Unit Test
VCS	Version Control System

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

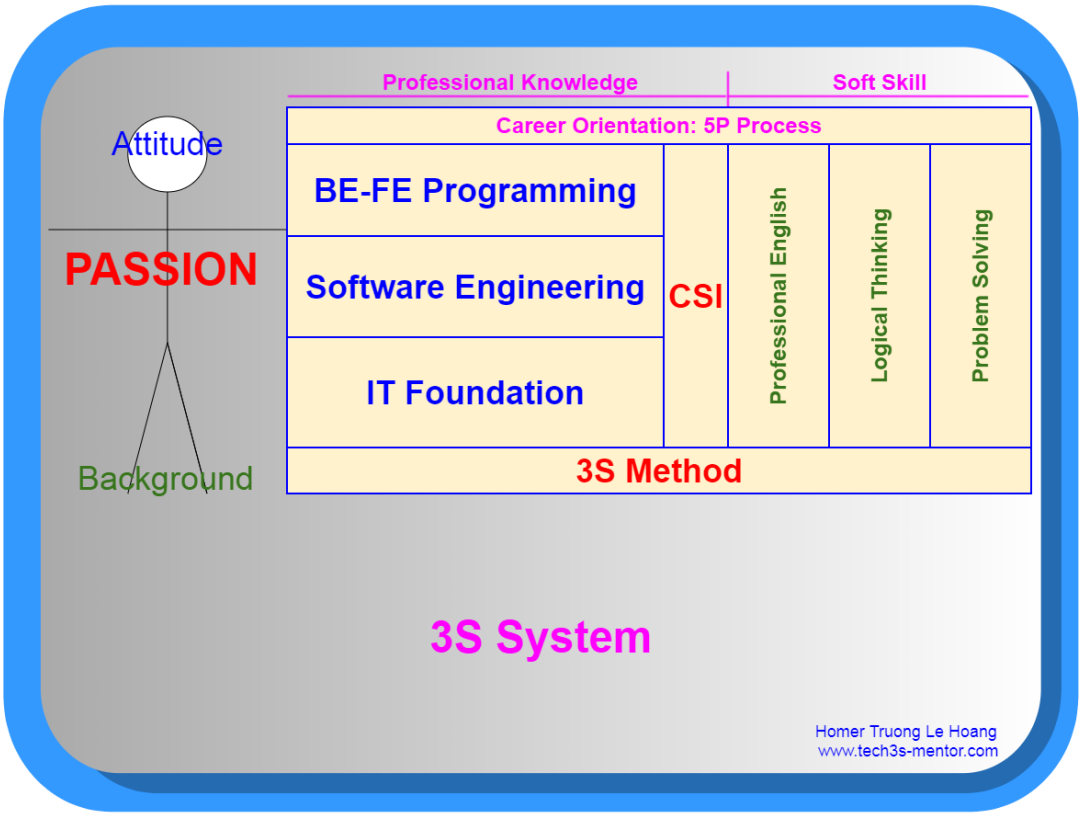
Tính hợp lý	**Logic**
Suy nghĩ luận lý	**Logical Thinking**
Tính thích ứng	**Adaptability**
Tinh thần làm chủ	**Enterpreneurial Spirit**
Nguồn năng lượng	**Energy**
Ngôn ngữ lập trình	**Programming Language**
Mạng nội bộ	**Local Area Network – LAN**
Mạng toàn cầu	**Internet**
Giải pháp trong nhà	**In-house Solution**
Giải pháp đám mây	**Cloud Solution**
Tính toàn vẹn dữ liệu	**Data Integrity**
Kiến thức lĩnh vực	**Domain Knowledge**
Cấu trúc máy tính	**Computer Structure**
Vòng đời phát triển phần mềm	**Software Development Life Cycle – SDLC**
Phương pháp thác nước	**Waterfall Methodology**
Phương pháp nhanh & lặp	**Agile Methodology**
Yêu cầu chức năng	**Functional Requirement – FR**
Yêu cầu phi chức năng	**Non-Functional Requirement – NFR**
Luồng nghiệp vụ	**Business Flow**
Qui luật nghiệp vụ	**Business Rule**
Lược đồ thiết kế	**Design Diagram**
Góc nhìn luận lý	**Logical View**
Góc nhìn phát triển	**Physical View**
Góc nhìn qui trình	**Process View**
Góc nhìn triển khai	**Deployment View**
Tình huống	**Scenario**
Nhóm công nghệ	**Technology Stack**
Khung hệ thống	**System Skeleton**
Kiểm tra đơn vị	**Unit Test – UT**
Mã nguồn	**Source Code**
Bộ mã nguồn	**Souce Code Repository**
Hệ quản lý phiên bản	**Version Control System – VCS**
Sản phẩm công nghệ	**Tech Product**
Thiết bị di động	**Mobile Device**
Điện thoại thông minh	**Smart Phone**
Mẫu thiết kế	**Design Pattern**
Cú pháp	**Syntax**
Thư viện	**Library**
Khung làm việc	**Framework – FW**
Danh sách	**List**
Tập hợp	**Set**
Cây	**Tree**
Nút gốc	**Root**
Nút trung gian	**Intermediate Node**
Nút lá	**Leaf**
Ánh xạ	**Map**
Khóa	**Key**
Giá trị	**Value**

Ứng dụng đa trang	**Multi-Page Application – MPA**
Ứng dụng đơn trang	**Single Page Application – SPA**
Trải nghiệm người dùng	**User Experience – UX**
Lập trình thủ tục	**Procedure Programming**
Lập trình hướng đối tượng	**Object-Oriented Programming – OOP**
Thuộc tính	**Property**
Thủ tục	**Procedure**
Phương thức	**Method**
Tính bao đóng	**Encapsulation**
Tính thừa kế	**Inheritance**
Tính đa hình	**Polymorphism**
Lập trình khía cạnh	**Aspect-Oriented Programming – AOP**
Lập trình hàm	**Functional Programming**
Môi trường phát triển tích hợp	**Integrated Development Environment – IDE**
Nguyên nhân chính	**Root Cause**
Ưu/nhược điểm	**Pros/Cons**
Thử nghiệm ý tưởng	**Proof of Concept – POC**
Hiệu năng	**Performance**
Bộ nhớ đệm của trình duyệt	**Browser Cache**
Tiểu tiến trình	**Thread**
Đồng bộ	**Synchronous**
Bất đồng bộ	**Asynchronous**
Thất thoát bộ nhớ	**Memory Leak**
Vấn đề	**Problem**
Dữ kiện	**Fact**
Kiến thức	**Knowledge**
Kinh nghiệm	**Experience**
Kết luận	**Conclusion**
Giải pháp	**Solution**
Có năng suất	**Productivity**
Ngoài giờ	**Over Time – OT**

Lời tổng kết

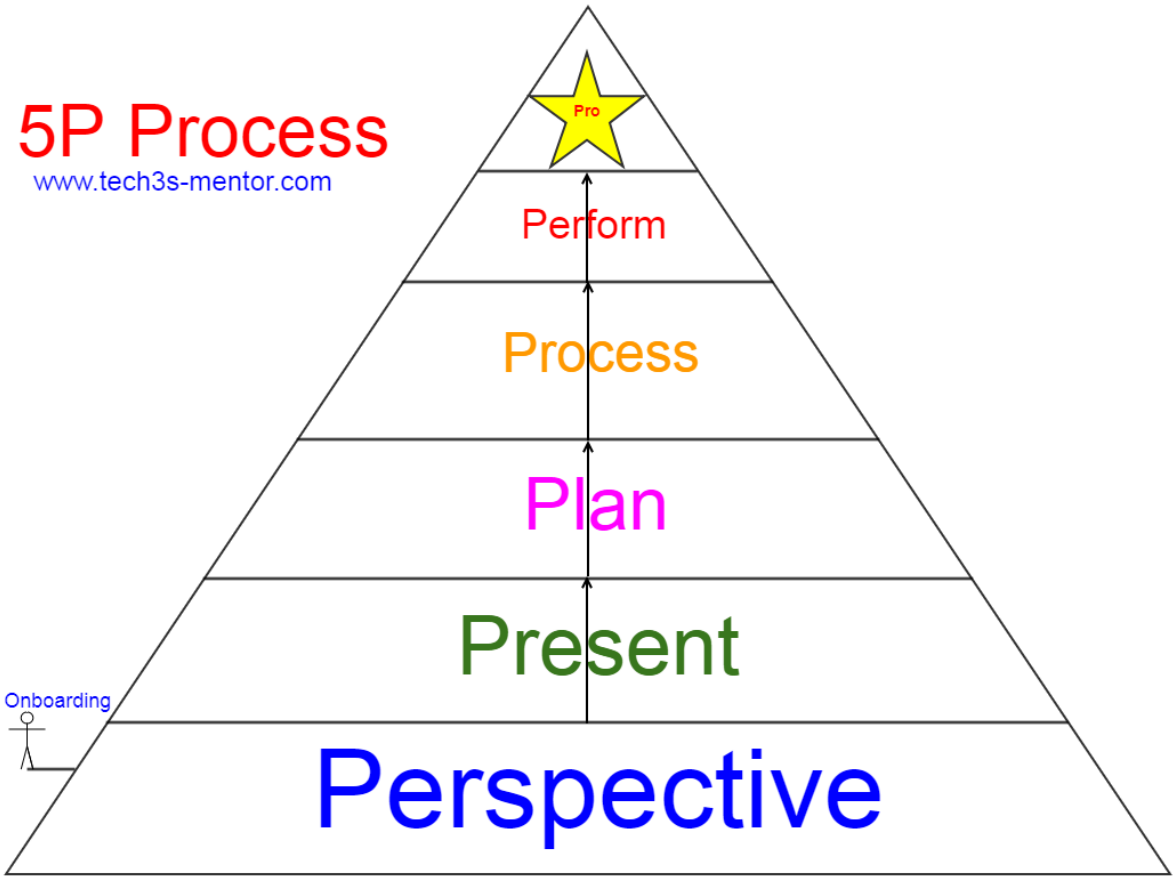
Cuốn cẩm nang "Phát triển phần mềm chuyên nghiệp" này đã chia sẻ với bạn kiến thức tổng hợp và kinh nghiệm của tôi để trở thành nhà phát triển phần mềm chuyên nghiệp.

Đó là Yếu tố con người, Kiến thức chuyên môn và Kỹ năng mềm, được thể hiện một cách trực quan qua Hệ thống 3S được đúc kết từ 15 năm trải nghiệm trong ngành IT của tôi trong hình sau:

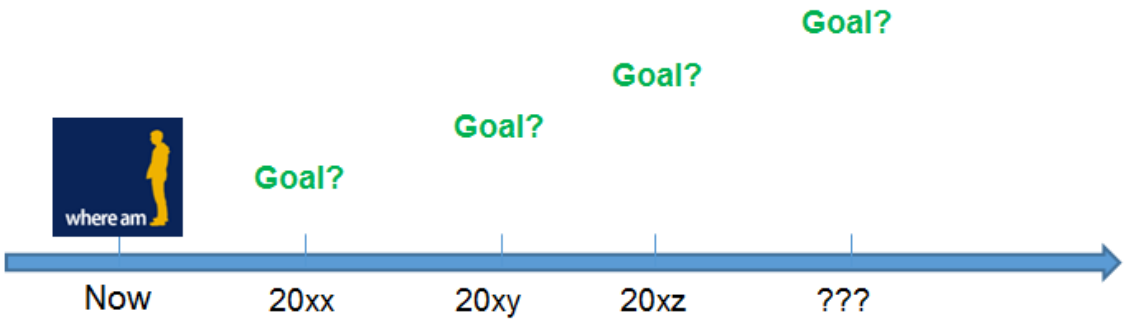


Với Kiến thức chuyên môn và Kỹ năng mềm đã được chia sẻ thì bạn đã biết những gì cần làm để trở thành nhà phát triển phần mềm chuyên nghiệp.

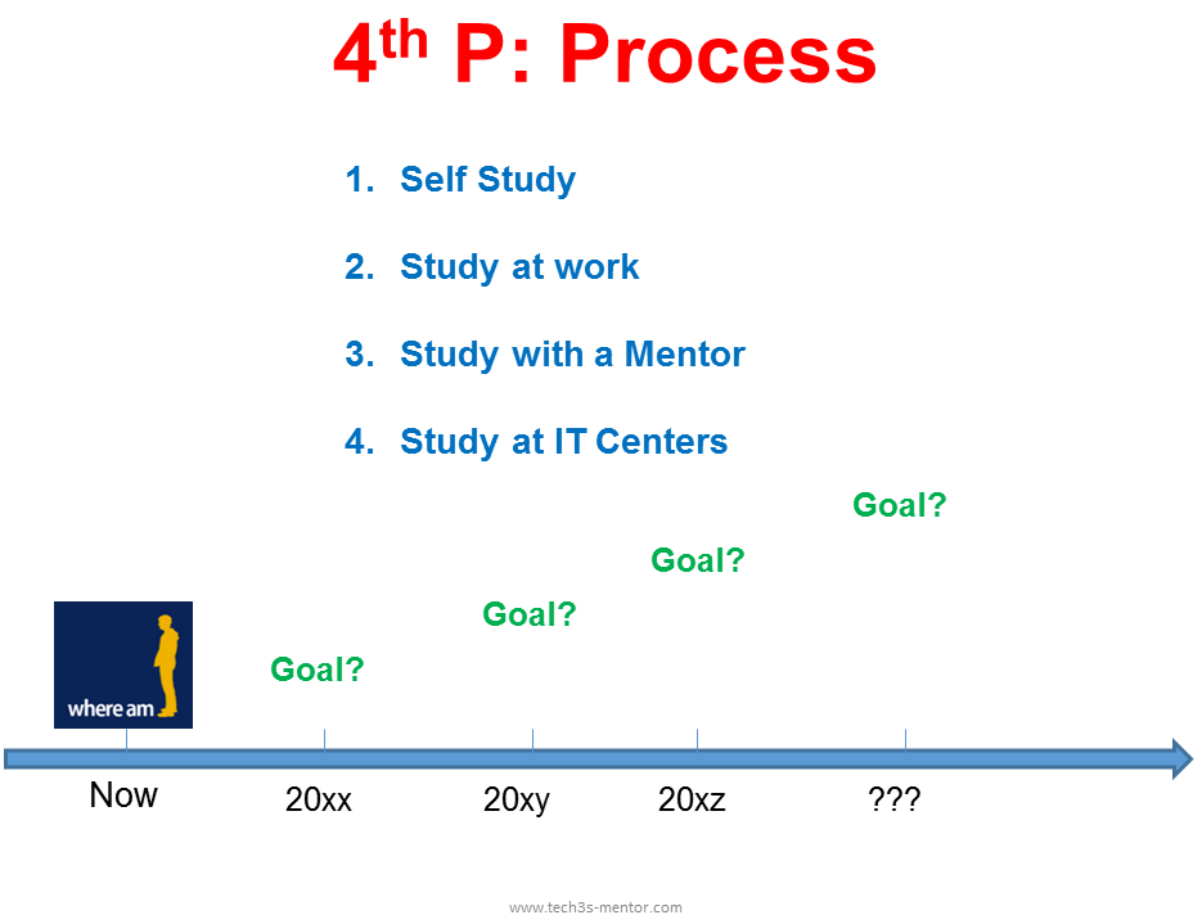
Bước tiếp theo là bạn nên lập kế hoạch nghề nghiệp (Career Orientation) cho bạn sử dụng **qui trình 5P (5P Process)** của tôi.



- **Perspective**: tầm nhìn về nghề của bạn giúp bạn biết được bạn sẽ phải trải qua những nấc thang nào để lên đến đỉnh cao nghề của bạn
  - Fresher SD
  - SD
  - Senior SD
  - SA / Manager (theo hướng quản lý)
  - CTO / Director
- **Present**: trạng thái hiện tại của bạn
- **Plan**: lập kế hoạch chi tiết cho bạn với những mốc thời gian cụ thể tính từ bây giờ và mục tiêu tại các mốc thời gian đó



- **Process**: xác định qui trình thực hiện



- **Perform**: cuối cùng là thực hiện kế hoạch một cách quyết tâm và kỉ luật nhất để đạt các mục tiêu đề ra.

Hi vọng cuốn cẩm nang này sẽ soi đường cho bạn tiến nhanh và chắc trên con đường trở thành nhà phát triển phần mềm chuyên nghiệp ☺

Mọi ý kiến đóng góp của bạn cho cuốn cẩm nang tốt hơn để phục vụ càng nhiều bạn đam mê IT hơn vui lòng gửi về homertruong66@gmail.com.
Xin cảm ơn các bạn!

Các bạn có thắc mắc gì muốn hỏi thì hãy kết nối và theo dõi Facebook dành cho IT 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