



NỀN TẢNG VỮNG CHẮC - KHỞI SẮC TƯƠNG LAI



HOMER TRƯƠNG LÊ HOÀNG



Homer Truong Le Hoang

Vietnam Top Software Professional Mentor

Canada Master of Information Technology (IT)

Work 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>

Hệ thống 3S cho dân IT

Lời mở đầu 9

A. Human (Con người) 12

1) <i>PASSION (ĐAM MÊ)</i>	12
2) <i>Attitude (Tư duy)</i>	13
a. <i>Background (Nền tảng)</i>	13
b. <i>Can-do (Có thể làm)</i>	13
c. <i>Logic (Sự hợp lý)</i>	14
d. <i>Eager-to-learn Spirit (Tinh thần ham học hỏi)</i>	14
e. <i>Entrepreneur Spirit (Tinh thần làm chủ)</i>	14
f. <i>Goal (Mục tiêu)</i>	14
g. <i>Environment (Môi trường)</i>	15
h. <i>Mentor</i>	15
3) <i>3S Method (Phương pháp 3S)</i>	16

B. Professional Knowledge (Kiến thức chuyên ngành) 17

1) <i>Computer Structure – CST (Cấu trúc máy tính)</i>	19
a. <i>Computer History (Lịch sử máy tính)</i>	19
b. <i>Computer Types (Các loại máy tính)</i>	19
c. <i>Computer Devices (Các thiết bị máy tính)</i>	19
d. <i>Computer-User Interaction (Tương tác người dùng với máy tính)</i>	20
e. <i>Central Processing Unit – CPU (Bộ xử lý trung tâm)</i>	21
f. <i>Random Access Memory – RAM (Bộ nhớ truy cập ngẫu nhiên)</i>	21

g.	Read Only Memory – ROM (Bộ nhớ chỉ đọc).....	21
h.	Hard Disk – HD (Đĩa cứng)	21
i.	Unit of Measurement (Đơn vị đo lường).....	22
j.	File (Tập tin)	22
k.	Self-Experience	23
2)	Operating System – OS (Hệ điều hành)	24
a.	OS History (Lịch sử OS)	24
b.	Popular OSs (Các OS phổ biến)	24
c.	Processor Management (Quản lý CPU)	25
d.	Process Management (Quản lý tiến trình)	26
e.	Memory Management (Quản lý bộ nhớ).....	27
f.	Disk Management (Quản lý đĩa cứng)	28
g.	I/O Management (Quản lý nhập xuất)	29
h.	Self-Experience	30
3)	Software – SW (Phần mềm)	31
a.	Closed-Source Software – CSSW (Phần mềm nguồn đóng).....	32
b.	Open-Source Software – OSSW (Phần mềm nguồn mở)	32
c.	Software as a Service – SaaS (Phần mềm là dịch vụ)	33
d.	Self-Experience	33
4)	Database – DB (Cơ sở dữ liệu)	34
a.	Relational Database (Cơ sở dữ liệu quan hệ)	35
b.	Entity Relationship Diagram – ERD (Mô hình quan hệ thực thể).....	35
c.	Database (De)Normalization ((Phá) Chuẩn hóa DB).....	36
d.	Structured Query Language – SQL (Ngôn ngữ truy vấn cấu trúc)	36
e.	Data Definition Language – DDL (Ngôn ngữ định nghĩa dữ liệu)	37
f.	Data Manipulation Language – DML (Ngôn ngữ xử lý dữ liệu).....	37
g.	Data Control Language – DCL (Ngôn ngữ điều khiển truy cập dữ liệu)	37
h.	Database Transaction (Giao dịch DB)	38
i.	Transaction Control Language – TCL (Ngôn ngữ điều khiển giao dịch)	38
j.	NoSQL Database (Cơ sở dữ liệu NoSQL)	38
k.	Database Management System – DBMS (Hệ quản trị DB)	39
l.	Database Selection (Lựa chọn DB)	39
m.	Self-Experience	40
5)	Computer Network – CNE (Mạng máy tính)	41
a.	CNE History (Lịch sử mạng máy tính)	42

b.	CNE Types (Các loại mạng máy tính)	42
c.	CNE Components (Các thành phần trong mạng máy tính)	42
d.	OSI Model (Mô hình OSI)	43
e.	TCP/IP Model (Mô hình TCP/IP).....	43
f.	IP Address (Địa chỉ IP)	44
g.	Subnet (Mạng con)	45
h.	Domain Name (Tên miền).....	45
i.	Protocol (Giao thức)	46
j.	Networking Application (Ứng dụng có nối mạng)	46
k.	Web Application (Ứng dụng Web).....	47
l.	Service Port (Cổng dịch vụ).....	47
m.	Popular Internet Services (Các dịch vụ phổ biến trên Internet)	48
n.	Self-Experience	48
6)	Programming (Lập trình)	50
a.	Programming Language – PL (Ngôn ngữ lập trình).....	50
b.	PL History (Lịch sử ngôn ngữ lập trình).....	50
c.	Translating Program – TPG (Chương trình dịch)	51
d.	Structure of a Program (Cấu trúc của 1 chương trình)	52
e.	Programming Concepts (Các khái niệm trong lập trình).....	52
f.	Procedure – PROC (Thủ tục).....	54
g.	PROC-related Concepts (Các khái niệm liên quan đến PROC).....	55
h.	Programming Model – PMO (Mô hình lập trình)	56
i.	Procedure Programming – PP (Lập trình thủ tục)	57
j.	Object-Oriented Programming – OOP (Lập trình hướng đối tượng)	57
k.	Aspect-Oriented Programming – AOP (Lập trình hướng khía cạnh)	62
l.	Functional Programming – FP (Lập trình hàm)	63
m.	Data Structure – DST & Algorithms (Cấu trúc dữ liệu & Giải thuật).....	63
n.	Design Patterns (Các mẫu thiết kế)	65
o.	Library – Lib (Thư viện) & Framework – FW (Bộ khung)	66
p.	Debugging (Tìm lỗi)	66
q.	Code Quality (Chất lượng mã).....	66
r.	Version Control System – VCS (Hệ điều khiển phiên bản)	69
s.	Git Standard (Chuẩn Git)	70
t.	Code Conflict (Xung đột mã).....	70
u.	Self-Experience	71
7)	Artificial Intelligence – AI (Trí tuệ nhân tạo)	74

a.	AI Fundamental Formula (Công thức nền tảng của AI)	74
b.	AI Applications (Các ứng dụng AI)	75
c.	Self-Experience	75
8)	Data Science – DSC (Khoa học dữ liệu)	76
a.	Data Analytics (Phân tích dữ liệu)	76
b.	Data Mining (Khai thác dữ liệu).....	77
c.	Data Warehouse – DW (Kho dữ liệu)	81
d.	Self-Experience	82
9)	Software Engineering – SWEN (Công nghệ phần mềm)	83
a.	Software Development Life Cycle – SDLC (Vòng đời phát triển SW).....	83
b.	Software Development Process (Qui trình phát triển phần mềm).....	83
c.	“4+1 Views” Model (Mô hình “4+1 Views”).....	84
d.	Phân tích (Analyze)	85
e.	Thiết kế (Design)	85
f.	Hiện thực (Implement).....	85
g.	Kiểm thử (Test).....	86
h.	Triển khai (Deploy).....	86
i.	Bảo trì (Maintain)	86
j.	Self-Experience	86
C.	Soft Skills (Kỹ năng mềm)	87
1)	Professional English (Tiếng Anh làm việc)	87
a.	Nature of Word (Bản chất của từ).....	88
b.	Function of Word (Chức năng của từ)	89
c.	Compound Noun (Danh từ ghép).....	89
d.	Sentence (Câu)	91
e.	Tense (Thì)	94
f.	Reading Skill (Kỹ năng Đọc)	95
g.	Writing Skill (Kỹ năng Viết).....	96
h.	Listening Skill (Kỹ năng Nghe).....	96
i.	Speaking Skill (Kỹ năng Nói).....	97
j.	Popular Misunderstandings (Các hiểu lầm phổ biến)	98
k.	Self-Experience	99
2)	Logical Thinking (Suy nghĩ luận lý)	100
a.	Introduction	100
b.	Self-Experience	101

3)	<i>Problem Solving (Giải quyết vấn đề)</i>	102
a.	Introduction	102
b.	Self-Experience	102
4)	<i>Time Management (Quản lý thời gian)</i>	103
a.	Introduction	103
b.	Elsenhower's Matrix (Ma trận của Elsenhower)	104
c.	Self-Experience	104
5)	<i>Team Work (Làm việc nhóm)</i>	105
a.	Introduction	105
b.	Self-Experience	106
6)	<i>Presentation & Communication (Thuyết trình & Giao tiếp)</i>	107
a.	Introduction	107
b.	Self-Experience	107

D. Advanced Knowledge (Kiến thức nâng cao) 108

1)	<i>Software Architecture – SWAR (Kiến trúc phần mềm)</i>	109
a.	Client-Server Model (Mô hình Khách-Chủ)	109
b.	Desktop Application Architecture (Kiến trúc ứng dụng Desktop)	109
c.	Model-View-Controller – MVC Pattern (Mẫu MVC)	109
d.	Web Application Architecture (Kiến trúc ứng dụng Web)	110
e.	Logical Separation (Layer) vs Physical Separation (Tier)	111
f.	Multiple Page Application – MPA (Ứng dụng đa trang)	112
g.	Server-side MVC Pattern (Mẫu MVC phía Server)	112
h.	Single Page Application – SPA (Ứng dụng đơn trang)	112
i.	System Integration (Tích hợp hệ thống)	113
j.	Monolithic Application vs Micro-services Architecture	113
k.	Self-Experience	114
2)	<i>Backend Programming – BEP (Lập trình phía Backend)</i>	116
a.	Application Programming Interface – API (Giao tiếp lập trình ứng dụng)	116
b.	REpresentational State Transfer – REST Architeture (Kiến trúc REST)	116
c.	JSON Processing (Xử lý JSON)	117
d.	Object Mapping (Ánh xạ đối tượng)	117
e.	Interceptor (Bộ chặn) & Web Filter (Bộ lọc Web)	118
f.	Inversion of Control – IoC (Đảo ngược điều khiển)	119
g.	Event-Driven Programming (Lập trình dựa vào sự kiện)	119
h.	Dependency Injection – DI (Tiêm sự phụ thuộc)	119

i.	User Authentication (Định danh người dùng)	120
j.	User Authorization (Phân quyền người dùng).....	121
k.	System Logging (Lưu vết hệ thống)	121
l.	Data Validation (Xác nhận dữ liệu)	122
m.	Business Transaction (Giao dịch nghiệp vụ)	122
n.	Object Relational Mapping – ORM (Ánh xạ quan hệ & đối tượng).....	122
o.	Data Search/Sorting/Paging (Tìm kiếm/Sắp xếp/Phân trang dữ liệu)	123
p.	Error Handling (Xử lý lỗi)	123
q.	Concurrency Handling (Xử lý đồng thời).....	124
r.	Asynchronous Processing (Xử lý bất đồng bộ)	125
s.	Resource Pooling Configuration (Threads, DB Connections...)	125
t.	Big File Downloading Handling (Xử lý việc tải file lớn).....	125
u.	Data Caching (Lưu dữ liệu tạm thời)	126
v.	Internalization – I18n (Đa ngôn ngữ).....	126
w.	External Integration: SMS, Payment, Mass Emails, AWS...	126
x.	Unit Test (Kiểm tra đơn vị).....	126
y.	Self-Experience	127
3)	Frontend Programming – FEP (Lập trình phía Frontend)	128
a.	User Interface – UI (Giao diện người dùng).....	128
b.	Hyper Text Markup Language – HTML	129
c.	Cascading Style Sheet – CSS	129
d.	Javascript	130
e.	Event Handling (Xử lý sự kiện).....	130
f.	Event Bubbling (Chuyển tiếp sự kiện)	130
g.	Client-side MVC Pattern (Mẫu MVC phía client).....	131
h.	Flux Architecture (Kiến trúc Flux)	132
i.	Redux Library (Thư viện Redux)	133
j.	UI Transition (Chuyển đổi giao diện)	133
k.	UI Data binding/unbinding (Nạp vào/Lấy ra dữ liệu trên UI)	134
l.	UI Paging Bar (Thanh phân trang UI).....	134
m.	Error Handling (Xử lý lỗi).....	134
n.	UI Routing (Định tuyến UI)	134
o.	Browsing History (Lịch sử duyệt Web)	135
p.	State Management (Quản lý trạng thái): UI State vs App State	135
q.	Asynchronous JavaScript and XML – AJAX Request	135
r.	Http Client (Thành phần truy cập dịch vụ Web)	136

s.	User eXperience – UX (Trải nghiệm người dùng)	136
t.	User Authentication (Định danh người dùng).....	136
u.	User Authorization (Phân quyền người dùng)	136
v.	Internationalization – I18n (Đa ngôn ngữ)	136
w.	Unit Test (Kiểm tra đơn vị)	137
x.	Self-Experience	137
4)	Non-Functional Requirement – NFR (Yêu cầu phi chức năng)	138
a.	System Performance (Hiệu năng hệ thống)	138
b.	System Availability (Độ sẵn sàng hệ thống).....	140
c.	System Scalability (Việc cung cấp hệ thống)	141
d.	System Security (Bảo mật hệ thống).....	141
e.	Self-Experience	141
5)	Reverse Engineering – REN (Kỹ thuật học ngược).....	142
a.	REN Process (Qui trình học ngược).....	142
b.	Self-Experience	143
6)	CSI Formula (Công thức CSI).....	144
a.	Concept – Spec – Impl (Khái niệm – Đặc tả – Hiện thực)	144
b.	Self-Experience	146
	Phụ Lục.....	148
1)	Abbreviation (Các từ viết tắt trong bài)	148
2)	Popular English Terms (Các thuật ngữ English thông dụng).....	151
3)	Software Development Toolkit (Bộ công cụ phát triển phần mềm) ..	164
	Lời tổng kết	165

Lời mở đầu

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

Trong quyển Cẩm nang “**Việc làm trong ngành IT**”, tôi đã giới thiệu tổng quan ngành IT và các nghề IT phổ biến với tính chất con người, kiến thức, và các kỹ năng cần có để theo đuổi từng nghề nhằm giúp các bạn trẻ ĐAM MÊ IT có thể xác định nghề IT nào phù hợp cho mình nhất. Nếu bạn chưa đọc cuốn cẩm nang trên của tôi thì hãy lên Facebook Fanpage TECH3S tải về đọc nhé.

Cho dù bạn làm nghề IT nào thì cũng cần nắm vững chắc những kiến thức nền tảng (**Computer Structure, Operating System, Software, Database, and Computer Network**) và những kỹ năng mềm (**English, Logical Thinking, Problem Solving, Time Management, Teamwork, Presentation, and Communication**), sau đó mới đến những kiến thức & kỹ năng chuyên biệt cho từng nghề IT (**Programming, Artificial Intelligence, Data Science, and Software Engineering**).

Qua thực tiễn trải nghiệm ở Việt Nam, tôi thấy phần lớn các bạn trẻ Việt Nam sau khi học xong IT thì chưa có được kiến thức nền tảng IT vững chắc, chưa đủ kỹ năng mềm để tham gia các dự án IT thực tế trong doanh nghiệp ngay khi ra trường.

Do đó, tôi đã quyết định viết cuốn cẩm nang “**Hệ thống 3S cho dân IT**” này để tổng hợp những tính chất một dân IT cần có, kiến thức nền tảng IT, và các kỹ năng mềm cần thiết phục vụ cho việc ĐI LÀM.

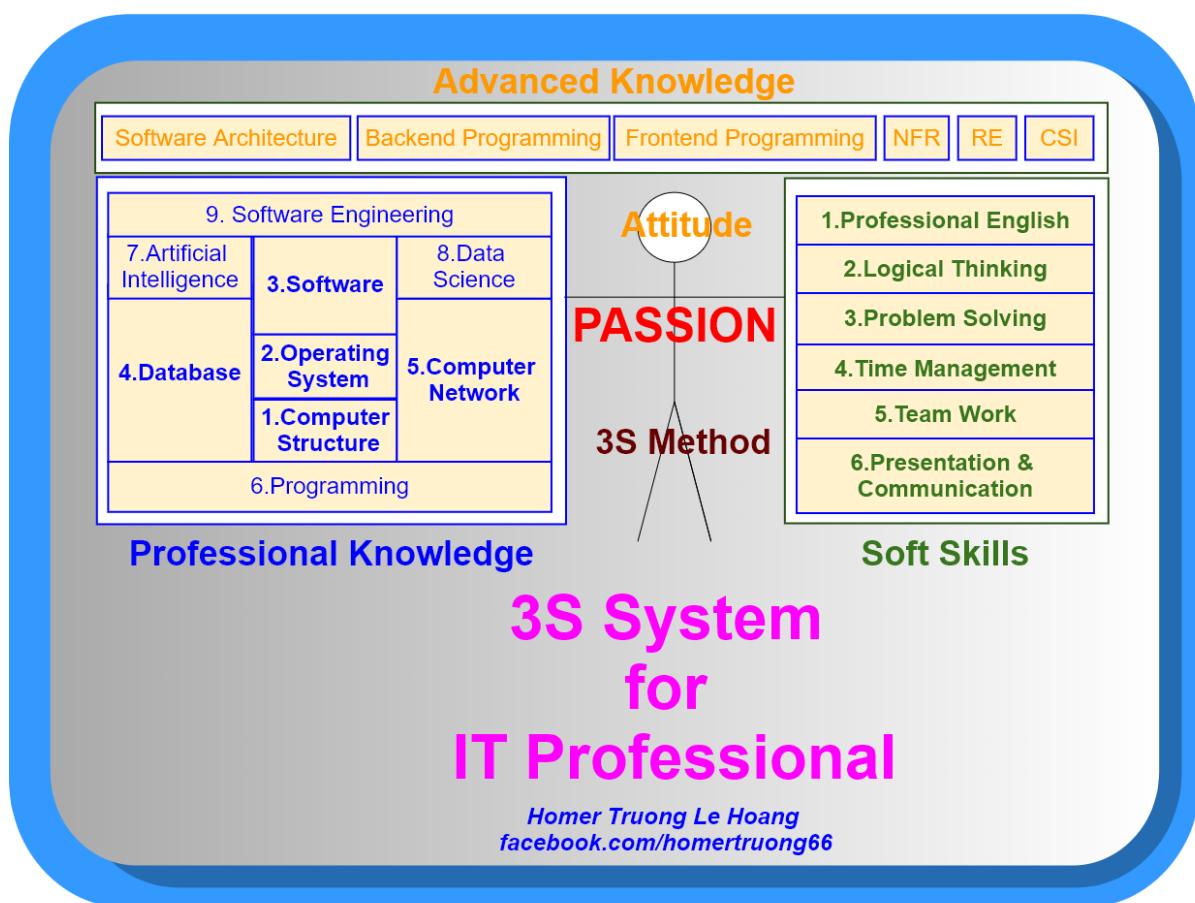
Ngoài ra, Hệ thống 3S cũng cung cấp một số kiến thức nâng cao giúp cho việc xây dựng các hệ thống IT tốt hơn:

1. Kiến trúc phần mềm (**Software Architecture**)
2. Lập trình phía Backend (**Backend – BE Programming**)
3. Lập trình phía Frontend (**Frontend – FE Programming**)
4. Yêu cầu phi chức năng (**Non-Functional Requirement – NFR**)
5. Kỹ thuật học ngược (**Reverse Engineering**)
6. Công thức CSI (**Concept-Specification-Implementation**) giúp tiếp cận công nghệ nhanh

Cuốn cẩm nang này sẽ hữu ích tùy tình huống của bạn hiện tại:

1. Nếu bạn định học IT thì cuốn cẩm nang này có thể xem như là bản đồ tổng thể về hành trình bạn sẽ đi và những gì bạn phải học & hành khi bạn quyết định theo ngành IT.
2. Nếu bạn đang học IT thì cuốn cẩm nang này sẽ giúp bạn củng cố kiến thức đang có & biết được những kiến thức cần học & hành tiếp theo.
3. Nếu bạn mới đi làm trong ngành IT vài năm thì cuốn cẩm nang này sẽ vừa giúp bạn củng cố kiến thức nền tảng & tích lũy thêm kiến thức nâng cao để làm việc hiệu quả hơn.

Hệ thống 3S bao gồm 4 phần được thể hiện tinh gọn qua hình sau:



1. Khu trung tâm đê cập về mặt con người: PHẢI có ĐAM MÊ (**PASSION**) mãnh liệt, PHẢI có Tư Duy (**Attitude**) đúng đắn và NÊN sử dụng phương pháp 3S (**3S Method**).
2. Nhánh bên trái nói đến kiến thức chuyên ngành (**Professional Knowledge**) gồm:
 - Kiến thức nền tảng (**Fundamental Knowledge**): màu xanh được in đậm
 - Kiến thức bổ sung (**Supplemental Knowledge**): màu xanh nhạt

3. Nhánh bên phải trình bày các kỹ năng mềm (**Soft Skill**) mà một người IT cần có để có thể làm việc nhóm hoặc làm việc độc lập một cách hiệu quả nhất.
4. Nhánh trên cùng là những phần kiến thức nâng cao (**Advanced Knowledge**) giúp xây dựng hệ thống IT tốt hơn cũng như giúp tiếp cận công nghệ nhanh hơn.

Ngành IT phát triển rất nhanh, vì thế những kiến thức tôt tổng hợp trong cuốn cẩm nang này hầu hết đều là cốt lõi nên sẽ ít thay đổi, và tôt cũng chỉ tập trung nhiều vào 2 khía cạnh **WHAT & WHY** (trong đó khía cạnh WHY là phần dân IT gặp nhiều khó khăn nhất) mà hầu như sẽ bỏ qua phần HOW rất là dài dòng phức tạp và thay đổi nhanh chóng.

Dựa trên WHAT & WHY, bạn phải luôn tìm tòi học hỏi & thực hành HOW để luôn nâng cao trình độ chuyên môn nhằm đáp ứng các yêu cầu ngày càng cao của công việc IT.

Một ghi chú quan trọng: nội dung trình bày về hệ thống 3S này được tổng hợp, chắt lọc, và tinh gọn nhằm PHỤC VỤ CHO NGƯỜI HỌC IT ĐI LÀM LÀ CHÍNH, những kiến thức thuộc dạng lý thuyết, hàn lâm, nghiên cứu khoa học được lược bỏ.

Cuốn cẩm nang có cấu trúc như sau:

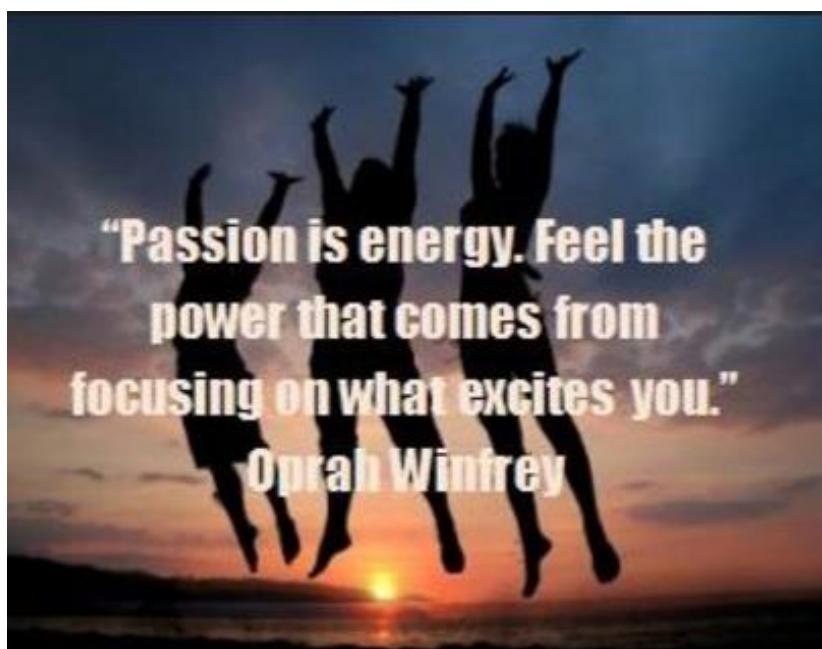
- **Phần A:** Khu trung tâm của hệ thống 3S đề cập đến yếu tố con người của dân IT.
- **Phần B:** Nhánh trái của hệ thống 3S tập trung vào kiến thức chuyên ngành.
- **Phần C:** Nhánh phải của hệ thống 3S trình bày về kỹ năng mềm.
- **Phần D:** Nhánh trên cùng của hệ thống 3S cung cấp thêm kiến thức nâng cao.

Chúng ta bắt đầu với khu trung tâm của hệ thống 3S...

A.Human (Con người)

1) PASSION (ĐAM MÊ)

Hãy để ý khi chúng ta ĐAM MÊ làm một việc gì đó thì sẽ có 1 nguồn năng lượng được sinh ra chạy khắp cơ thể giúp chúng ta tràn đầy cảm hứng và tập trung hoàn toàn tâm trí vào việc đang làm, đôi khi quên cả thời gian cũng những sự kiện khác đang diễn ra xung quanh; và kết quả đạt được thường rất tốt đẹp.



Ngược lại chúng ta làm việc gì đó mà không có đam mê, do bị ép buộc hay vì lý do gì khác mà không thể không làm thì tâm trí sẽ không ủng hộ chúng ta hoàn thành công việc này và hầu hết kết quả sau cùng sẽ 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 ngành IT đầy thách thức và áp lực dù không kém phần thú vị.

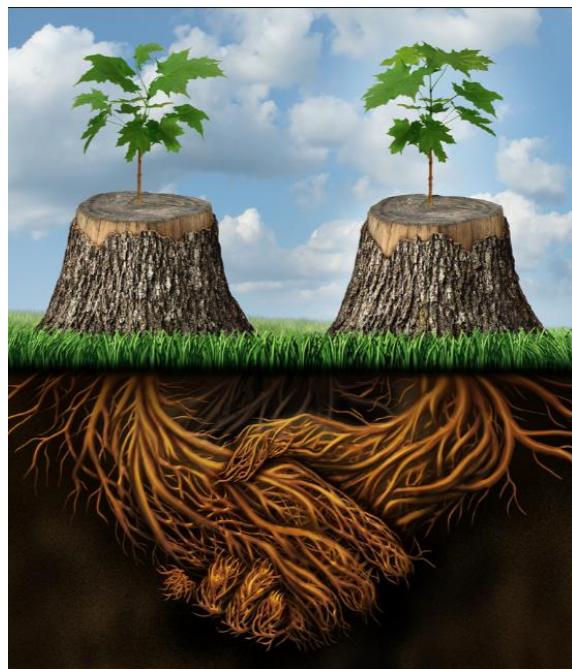
Cho nên, bạn hãy dừng lại một chút để xem xét lại bạn có thực sự đang theo ngành IT vì ĐAM MÊ hay vì lý do gì khác?

- Nếu câu trả lời là vì ĐAM MÊ thì xin chúc mừng bạn!
- Nếu bạn chưa có câu trả lời ngay thì hãy tạo cơ hội cho bạn thử sức với nghề IT.
- Nếu bạn đã học/làm nghề IT 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.

2) *Attitude (Tư duy)*

a. **Background (Nền tảng)**

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 bền vững hơn. Ví dụ thay vì chúng ta nhảy vô học ngay một ngôn ngữ lập trình (**Programming Language – PL**) cụ thể nào đó (như Java, C#, PHP, Python, Ruby...) thì chúng ta nên tìm hiểu trước căn bản của PL, rồi thực hành bằng một PL cụ thể để hiểu rõ các đặc điểm, tính chất cốt lõi của PL. Sau này chúng ta có nhu cầu phải chuyển qua học các PL khác sẽ nhanh hơn nhiều so với chúng ta học từng PL riêng rẽ.

b. **Can-do (Có thể làm)**

Chúng ta đang dự định làm một việc gì đó mà lúc nào cũng nghĩ việc này khó quá, sợ làm không được thì cơ thể sẽ sinh ra một lực vô hình chống lại hành động của chúng ta và khả năng cao là việc này sẽ thất bại.

Ngược lại, dù việc rất khó (khả thi chứ không viễn vông) nhưng lúc nào chúng ta cũng tin tưởng là sẽ làm được và luôn có gắng hết sức cũng như tâm trí để thực hiện thì cơ thể sẽ sản sinh ra một nguồn năng lượng tích cực (**Positive Energy**) giúp hoàn thành công việc hiệu quả một cách đáng kinh ngạc.

Cho nên, tư duy “**Có thể làm**” (“Can-do”) là cực kỳ quan trọng, đóng góp vào sự thành công của chúng ta ngay từ khi bắt đầu làm một việc gì đó.

c. Logic (Sự hợp lý)

Trong cuộc sống, mọi vấn đề trong mọi lĩnh vực luôn vận hành theo những sự hợp lý (**Logic**) nào đó. Hệ thống IT giúp giải quyết những vấn đề đó một cách tự động hóa thì cũng phải tuân theo những logic đó, vì thế những người xây dựng hệ thống IT cũng phải luôn suy nghĩ logic (**Logical Thinking**) thì mới tạo ra được những hệ thống IT tốt, hạn chế những lỗi sai logic.

d. Eager-to-learn Spirit (Tinh thần ham học hỏi)

Ngành IT luôn phát triển rất nhanh, cho nên tinh thần ham học hỏi (**Eager-to-learn Spirit**) và tính thích ứng cao (**High Adaptability**) để nắm bắt công nghệ mới đóng vai trò then chốt trong việc bắt kịp thời đại và làm ra những sản phẩm tốt nhất.

e. Entrepreneur Spirit (Tinh thần làm chủ)

Khi xây dựng 1 sản phẩm IT, chúng ta nên có tinh thần làm chủ (**Entrepreneur Spirit**) của sản phẩm dù có đang làm thuê. Chỉ khi có tinh thần làm chủ thì chúng ta 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. Rồi khi sản phẩm ngày càng tốt sẽ giúp công ty phát triển tốt lên thì chúng ta cũng được hưởng lợi cả về tinh thần và vật chất.

Ngược lại nếu chúng ta nghĩ làm công ăn lương thì chỉ làm cho xong những việc được giao rồi về nhà cho khỏe, khỏi suy nghĩ nhiều chi cho mệt óc thì 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.

f. Goal (Mục tiêu)

Ở mỗi giai đoạn phát triển sự nghiệp, chúng ta nên đặt ra mục tiêu dài hạn (**Long-term Goal**), trong đó sẽ có nhiều mục tiêu ngắn hạn (**Short-term Goal**), bởi vì mục tiêu giúp chúng ta có động lực và năng lực để phấn đấu tiến tới.

g. Environment (Môi trường)

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

h. Mentor

Tốt hơn nữa là chúng ta có được các **Mentor** hỗ trợ.



MENTOR follows and helps you a long the way.

Khái niệm Mentor không dịch được qua tiếng Việt vì không có từ tương ứng bên tiếng Việt, có thể hiểu nôm na Mentor là người đã trải qua những gì chúng ta đang/sẽ gặp, họ chỉ ra hướng đi phù hợp đồng thời đi theo sửa sai khi chúng ta học/làm việc thực tế.

Mentor có thể là những người bạn trong mạng lưới bạn bè của chúng ta hay những người mà chúng ta gặp trong các sự kiện hoặc những trường nhóm/đồng nghiệp trong môi trường làm việc thực tế.

3) 3S Method (*Phương pháp 3S*)

Tục ngữ có câu “Học đi đôi với hành”, nghĩa là chúng ta học mà không thực hành để sử dụng những kiến thức chúng ta đã tiếp thu được thì chúng ta sẽ mau quên.

Thật đúng là như vậy, với những gì tôi đã trải qua giờ nhìn lại tôi thấy là những kiến thức tôi đã học mà không có dịp thực hành thì tôi đều đã quên gần hết. Tôi cũng nhận thấy thêm một điều nữa là dù tôi có thực hành những kiến thức tôi đã học, nhưng thời gian và tần suất thực hành không đủ dài và nhiều để biến thành kỹ năng cho tôi thì sau một thời gian dài tôi cũng hầu như không còn xài được những kiến thức đó.

Với những trải nghiệm trên, tôi đã tổng hợp ra phương pháp 3S (3S Method) giúp cho việc học hiệu quả hơn, đó là **Study -> Self-Experience -> Skill**:

- **Study** (Học): khi học một công nghệ A nào đó thì chúng ta nên học theo thứ tự WHAT (nó là cái gì, nó có những tính năng gì...) -> WHY (tại sao nó ra đời/tồn tại) -> HOW (nó hoạt động thế nào, sử dụng nó ra sao...).
- **Self-Experience** (Tự trải nghiệm): học xong công nghệ A thì phải tự trải nghiệm rồi ngẫm nghĩ cách mình đã thực hành có phải là cách phù hợp nhất chưa, nếu chưa thì tìm cách làm tốt hơn rồi trải nghiệm tiếp... lặp lại quá trình này cho đến khi tìm được cách làm phù hợp nhất.
- **Skill** (Kỹ năng): trải nghiệm đủ lâu và nhiều để tạo ra kỹ năng cho chúng ta trên công nghệ A, gọi là kỹ năng thì khi chúng ta sử dụng công nghệ A thì chúng ta không cần phải suy nghĩ, làm một cách vô thức (**unconsciously**) mà kết quả luôn tốt đẹp.

Từ khi tổng hợp được 3S Method, tôi đã áp dụng thử thì thấy việc học của tôi hiệu quả hơn trước khá nhiều. Tôi tin là 3S Method đóng vai trò nền tảng trong việc học của dân IT.

Bạn nên áp dụng thử 3S Method xem hiệu quả thế nào nhé.

Vậy là chúng ta đã hiểu được 3 tính chất (PASSION, Attitude & 3S Method) thuộc yếu tố con người mà dân IT phải/cần có.

Thế thì dân IT phải có kiến thức gì để làm việc tốt trong ngành IT?

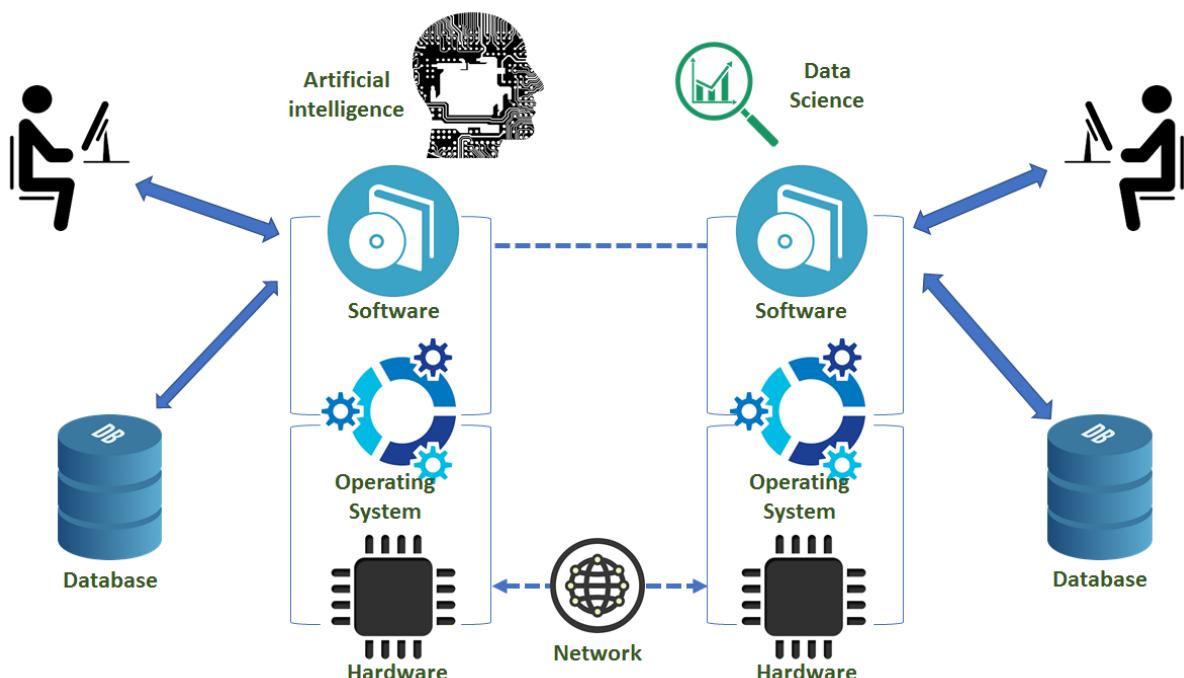
Câu trả lời sẽ nằm trong nhánh trái của hệ thống 3S được trình bày trong phần tiếp theo...

B. Professional Knowledge (Kiến thức chuyên ngành)

Dân IT làm bất cứ nghề nào thì cũng phải có kiến thức của 5 mảng nền tảng sau:

1. Cấu trúc máy tính (Computer Structure – CST)
2. Hệ điều hành (Operating System – OS)
3. Phần mềm (Software – SW)
4. Cơ sở dữ liệu (Database – DB)
5. Mạng máy tính (Computer Network – CNE)

Những mảng này tạo nên 1 hệ thống IT cơ bản trong hình dưới đây:



- Nhìn qua hình chúng ta thấy 2 cột dọc bên trái và bên phải là 2 máy tính (**Computer – CPT**) bao gồm 3 phần: Phần cứng (**Hardware – HW**), OS, và SW.
- Người dùng (**User**) tương tác với CPT bằng SW.
- SW thông qua OS điều khiển HW thực thi các lệnh của User.
- SW lưu dữ liệu (**Data**), khi User tương tác với SW qua thao tác tạo mới/đọc/chỉnh sửa/xóa (**Create/Read/Update/Delete – CRUD**), vào DB để lần sau nạp lên lại cho User tương tác tiếp.
- Trong quá trình SW S1 xử lý yêu cầu của User có thể cần dữ liệu được cung cấp bởi SW S2, S1 sẽ gọi S2 để lấy dữ liệu thông qua CNE.

Ví dụ thực tế: 1 User U1 sử dụng SW của ngân hàng VCB chuyển tiền cho 1 User U2 ở ngân hàng ACB. Giao dịch (**Transaction**) sẽ diễn ra như sau:

1. SW VCB trừ tiền tài khoản của U1 trên VCB rồi lưu xuống DB của VCB.
 2. SW VCB sẽ gọi SW ACB thông qua CNE để lấy thông tin về tài khoản U2 kiểm tra xem có đúng không, nếu ok thì SW VCB gửi yêu cầu qua SW ACB để tăng tiền tài khoản của U2 trên ACB.
 3. SW ACB tăng tiền tài khoản của U2 trên ACB rồi lưu xuống DB của ACB.
- > Cả 3 bước này được thực hiện tuần tự trong 1 Transaction, nếu một trong các bước thực hiện không thành công thì kết quả của những bước trước đó sẽ bị hủy.

Trong 5 chương tiếp theo chúng ta sẽ lần lượt tìm hiểu những mảng nền tảng này.

1) Computer Structure – CST (Cấu trúc máy tính)

a. Computer History (Lịch sử máy tính)

Máy tính được dùng để lưu trữ, truy cập và xử lý thông tin. Đây là một phát minh tuyệt vời của nhân loại trong thế kỷ 20 đã làm thay đổi thế giới rất nhiều cho đến ngày nay.

Máy tính đã trải qua một lịch sử khá dài với rất nhiều giai đoạn được miêu tả bằng dòng thời gian ở đây: <https://www.computerhistory.org/timeline/computers>.

b. Computer Types (Các loại máy tính)

Dựa trên quy mô sử dụng & kích thước, máy tính được chia thành một số loại sau:

- Máy tính lớn (**Main Frame**): dùng cho tổ chức lớn.
- Máy tính vừa (**Mini Frame**): tựa như máy tính lớn nhưng sử dụng ở mức độ thấp hơn.
- Siêu máy tính (**Super Computer**): 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.
- Máy tính cá nhân (**Personal Computer – PC**): dành cho User thông thường.
- Máy tính để bàn (**Desktop Computer**): được thiết kế với hộp máy nằm ngang (chứa màn hình) hoặc đứng.
- Máy tính xách tay (**Laptop Computer**): 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.
- Máy tính cầm tay (**Palmtop**): có kích thước rất nhỏ vừa trong lòng bàn tay.

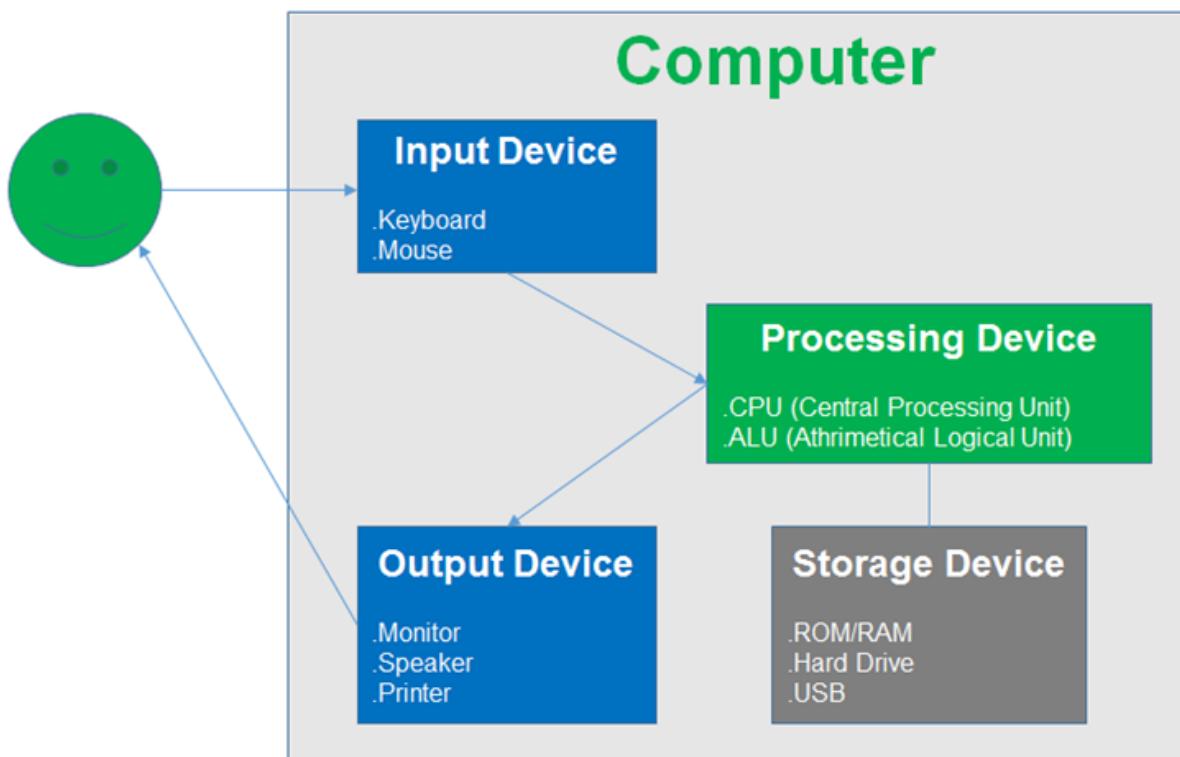
c. Computer Devices (Các thiết bị máy tính)



Một máy tính (**Computer – CPT**) gồm 4 loại thiết bị chuẩn sau:

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

d. Computer-User Interaction (Tương tác người dùng với máy tính)



- User ra lệnh cho CPT qua Input Device: bàn phím (**Keyboard**), chuột (**Mouse**)...
- Lệnh của User sẽ được Processing Device như bộ xử lý trung tâm (**Central Processing Unit – CPU**) và đơn vị xử lý luận lý số học (**Arithmetical Logical Unit – ALU**) xử lý rồi xuất kết quả qua Output Device như màn hình (**Monitor**), loa (**Speaker**), máy in (**Printer**)... để hiển thị cho User xem.
- Những lệnh cần đọc/lưu dữ liệu sẽ được thiết bị xử lý đọc từ/lưu vào 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 – HD**), **USB (Universal Serial Bus)**.

e. Central Processing Unit – CPU (Bộ xử lý trung tâm)

CPU điều khiển mọi hoạt động của CPT. CPU bao gồm 1 số thành phần quan trọng sau:

- **Control Unit – CU** (Đơn vị điều khiển): 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.
- **System Clock** (Mạch xung nhịp hệ thống): 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.
- **Arithmetical Logical Unit – ALU** (Đơn vị số học và luận lí): Đơn vị thực hiện phép toán số học và luận lí.
- **Micro Co-Processor – MCP** (Bộ đồng xử lí): Thành phần giúp CPU xử lí các phép toán dấu chấm động và các phép toán phức tạp khác.

f. Random Acess Memory – RAM (Bộ nhớ truy cập ngẫu nhiên)

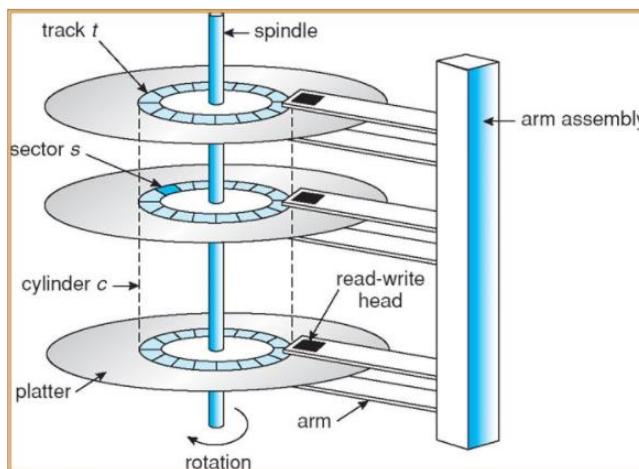
Bộ nhớ truy cập ngẫu nhiên (**Random Acess Memory – RAM**) là nơi chứa tạm chương trình và dữ liệu của chúng khi CPT đang hoạt động, toàn bộ nội dung trên RAM sẽ bị mất đi khi CPT tắt (shut down).

g. Read Only Memory – ROM (Bộ nhớ chỉ đọc)

Bộ nhớ chỉ đọc (**Read Only Memory – ROM**) là bộ nhớ chỉ cho đọc dữ liệu ra, không cho ghi dữ liệu vào, chứa các chương trình do hãng sản xuất cài đặt sẵn.

h. Hard Disk – HD (Đĩa cứng)

Cấu trúc vật lí (**Physical Structure**) của một HD gồm nhiều xy-lanh (**Cylinder – C**), trên mỗi C gồm các mặt đĩa (**Head – H**) và trên mỗi H được chia thành từng cung (**Sector – S**) có dung lượng là 512B -> Dung lượng đĩa (**HD Capacity**) = $C * H * S * 512$ (Bytes).



Trên OS Windows, cấu trúc luận lí (**Logical Structure**) của một HD có thể được chia thành nhiều phân khu (**Partition**) gồm 2 loại:

- **Primary Partition** (Phân khu chính): Là phân khu có thể chứa OS và chương trình để khởi động máy, mỗi HD có tối đa 4 phân khu chính, trong đó phải có ít nhất 1 phân khu chính ở trạng thái “**Active**” để khởi động máy.
- **Logical Partition** (Phân khu luận lý): Là các phân khu để chứa dữ liệu thuận, chúng được quản lý bởi phân khu mở rộng (**Extended Partition**) cũng là 1 Primary Partition.

i. Unit of Measurement (Đơn vị đo lường)

Hệ cơ số được sử dụng trong CPT là hệ nhị phân (**Binary System**) chỉ có số 0 và 1. Mỗi số 0 hoặc 1 gọi là 1 **bit (b)**.

Một **Byte (B)**, gồm 8 bit, là đơn vị chính được dùng trong CPT. Một Byte có giá trị thập phân từ 0-255, ví dụ: Byte 11111100 có giá trị là 252.

Các đơn vị tính lớn hơn Byte :

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

j. File (Tập tin)

Đơn vị cơ bản lưu trữ dữ liệu trong CPT là tập tin (**File**). Một File có nhiều tính chất, hai tính chất quan trọng của File là tên File (**File Name**) và thuộc tính File (**File Attribute**).

File Name gồm có 2 phần:

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

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

Đơn vị lớn hơn File là thư mục (**Directory**) dùng để gom nhóm các File liên quan, có thể xem như là danh mục cho File.

Ví dụ:

- Thư mục “Photo” chứa các File hình ảnh.
- Thư mục “Music” chứa các File nhạc.
- Thư mục “Project” chứa các File liên quan đến các dự án IT.

k. Self-Experience

- 1) Trình bày các thiết bị của một CPT và xác định các thiết bị này trên CPT của bạn.
- 2) Vẽ và trình bày lược đồ tương tác của User với CPT.
- 3) Trình bày các thành phần chính của một CPT và xác định các thành phần này trên CPT của bạn.
- 4) Xem tốc độ CPU trong CPT của bạn là bao nhiêu?
- 5) Xem dung lượng bộ nhớ RAM trong CPT của bạn bao nhiêu GB? Đổi sang MB, Bytes.
- 6) Xem dung lượng HD trong CPT của bạn bao nhiêu GB? Đổi sang MB, Bytes.
- 7) Nếu bạn đang xài OS Windows, xem HD trong CPT của bạn được phân chia thành các phân khu như thế nào?
- 8) Xem cấu trúc File & Directory trên HD trong CPT của bạn.

2) ***Operating System – OS (Hệ điều hành)***

a. **OS History (Lịch sử OS)**

Hệ điều hành (**Operating System – OS**) là một chương trình điều khiển mọi hoạt động của CPT, là thành phần trung gian giúp SW giao tiếp với HW.

OS gồm 3 thành phần chính:

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

Lịch sử phát triển của OS đã trải qua các giai đoạn sau:

- Xử lí theo lô (**Batch Processing**): mỗi lần nạp lên RAM một chương trình, chạy xong mới nạp tiếp chương trình khác.
- Xử lí đa chương trình (**Multi Programming**): 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.
- Xử lí đa nhiệm (**Multi Tasking**): 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.
- Xử lí song song (**Parallel Processing**): nhiều CPU cùng chạy trên cùng một CPT, xử lý nhiều chương trình song song cùng lúc.

b. **Popular OSs (Các OS phổ biến)**

Các OS phổ biến cho CPT bao gồm:

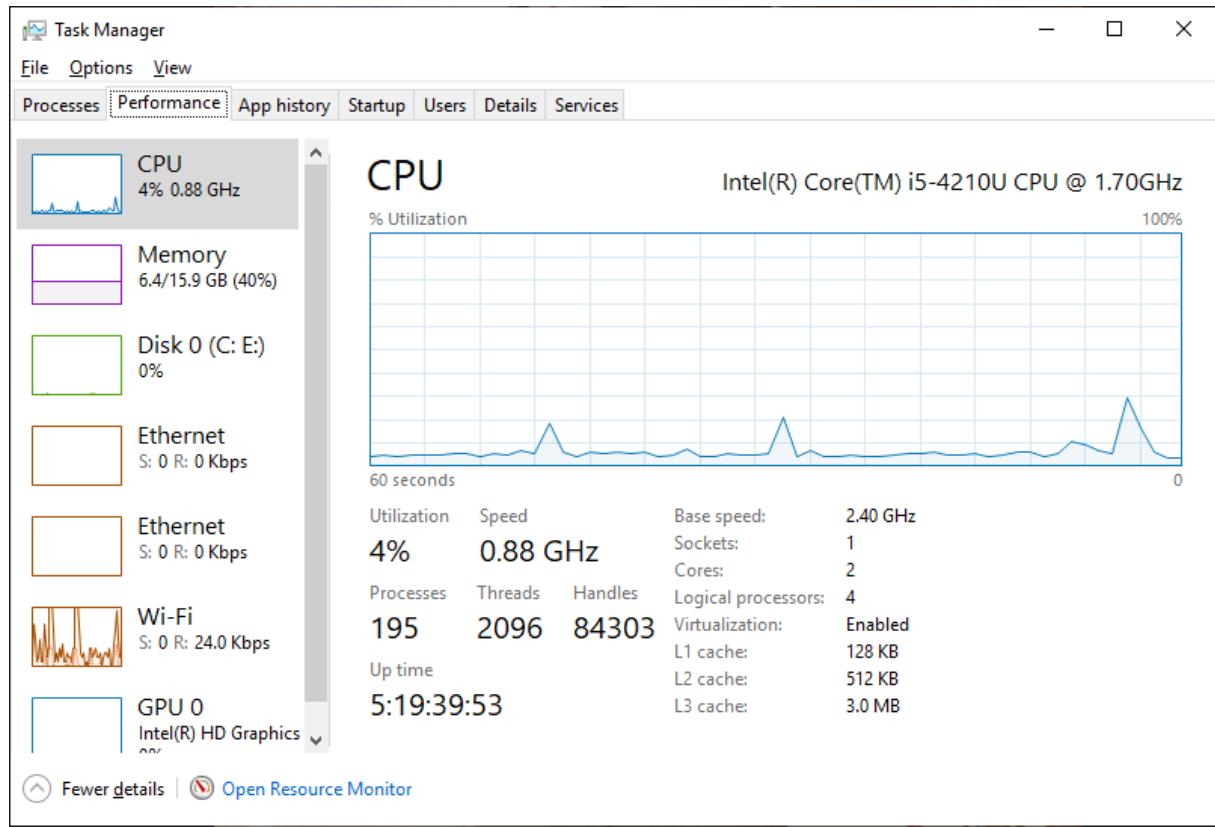
- **Microsoft Windows**: là OS của công ty Microsoft, có tính phí sử dụng.
- **Apple MacOS**: là OS của công ty Apple, miễn phí sử dụng khi dùng với các thiết bị của công ty Apple.
- **Linux OS**: là OS thuộc cộng đồng mã nguồn mở (**Open-Source Community**), miễn phí sử dụng. Linux OS có nhiều bản (**Linux Distribution**), phổ biến là Ubuntu, CentOS, Fedora, Manjaro.

Các OS phổ biến cho các thiết bị di động như điện thoại (**Phone**), máy tính bảng (**Tablet**):

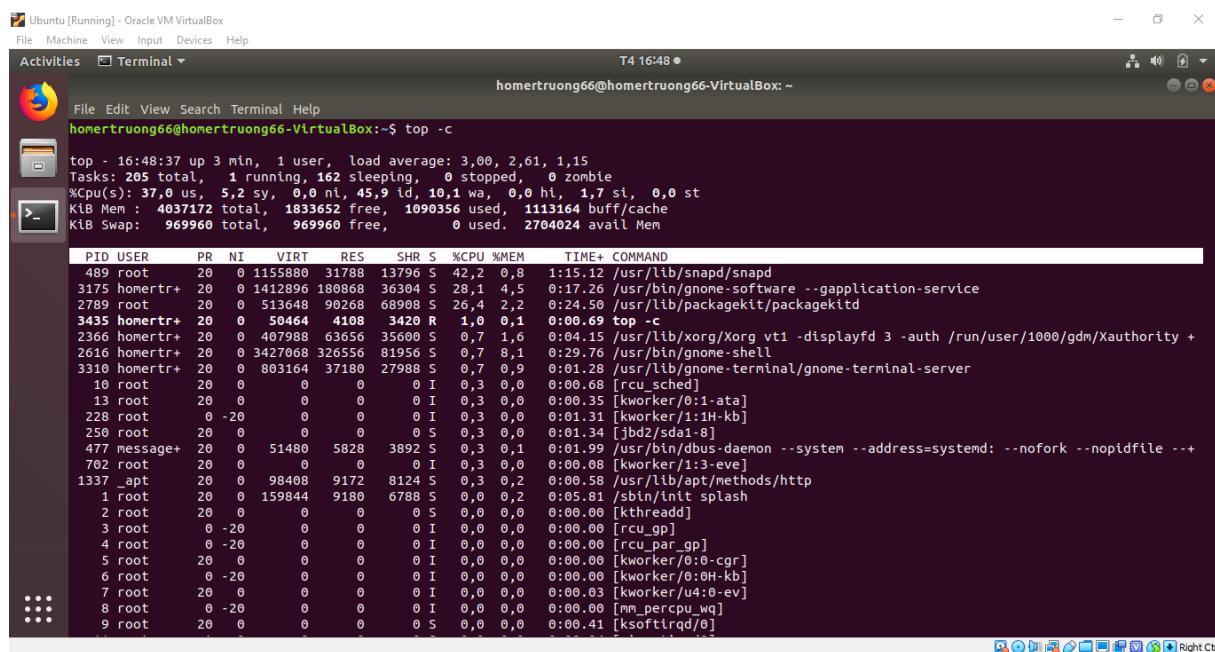
- **Apple iOS**: là OS của công ty Apple sử dụng trên “**iPhone**”, “**iPad**” của công ty Apple.
- **Google Android**: là OS của công ty Google sử dụng trên sản phẩm khác như điện thoại của công ty Google, Samsung...

c. Processor Management (Quản lý CPU)

Trên OS Windows, mở “Task Manager” để xem trạng thái CPU:



Trên OS Linux, chạy lệnh “**top -c**” để xem trạng thái CPU:



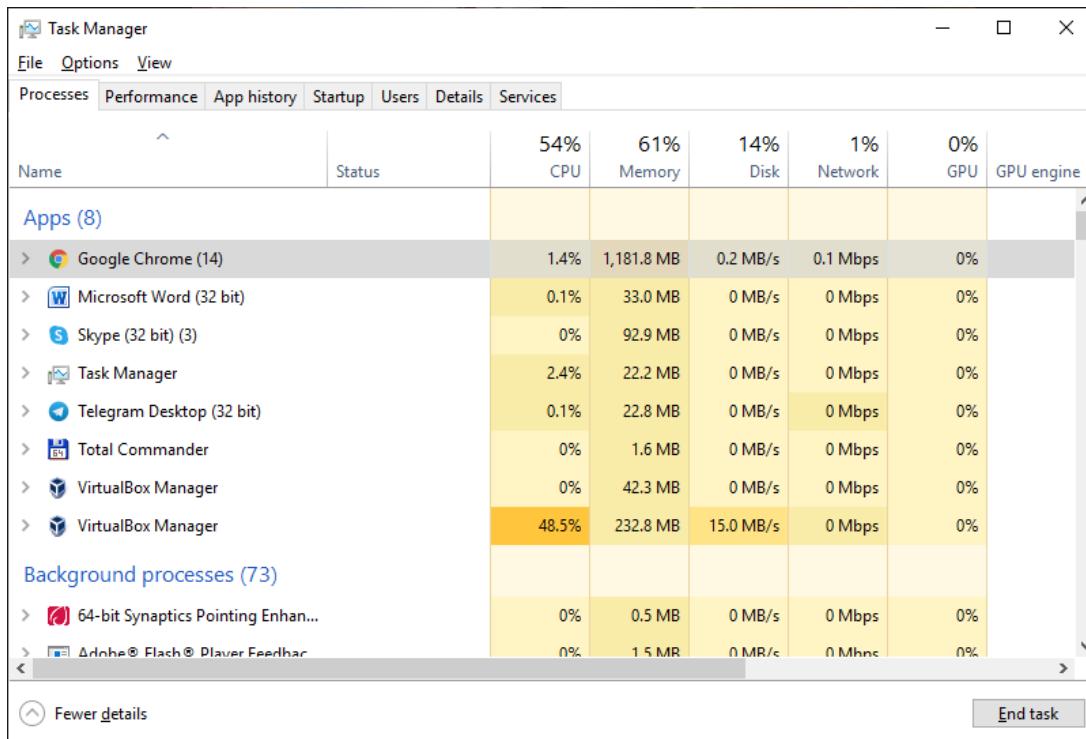
Nếu CPU đang sử dụng 100% hiệu năng thì có nghĩa CPT đang bị quá tải.

d. Process Management (Quản lý tiến trình)

Mỗi chương trình đang chạy trên OS thể hiện bằng một hoặc nhiều tiến trình (**Process**). Mỗi tiến trình có 3 trạng thái:

- Ready** (Sẵn sàng) : Process đang sẵn sàng để được chạy
- Running** (Đang chạy) : Process đang chạy
- Blocked** (Đứng hình) : Process đang ngưng để chờ nhập xuất (**Input/Output – IO**) hay chờ sự kiện tương tác phía User.

Trên OS Windows, mở “**Task Manager**” để xem các Process:



Trên OS Linux, chạy lệnh “**ps -aux**” để xem các Process:

```
vuhuong66@truonglehoang:~$ ps -aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      1  0.0  0.0 33240  3512 ?        Ss   2016  0:01 /sbin/init
root     153  0.0  0.0 19480  188 ?       S    2016  0:00 upstart-udev-bridge --daemon
root     186  0.0  0.0 49276  2620 ?       Ss   2016  0:00 /lib/systemd/systemd-udevd --daemon
syslog   268  0.0  0.6 256752 58128 ?      Ssl  2016  0:47 rsyslogd
root     289  0.0  0.0 15396  1688 ?      S    2016  0:00 upstart-socket-bridge --daemon
root     302  0.0  0.0 15280  252 ?       S    2016  0:00 upstart-file-bridge --daemon
root     306  0.0  0.0 10236  3816 ?      Ss   2016  1:00 dhclient -l -v -pf /run/dhclient.eth0
root     374  0.0  0.0 12792  1948 ?      Ss+  2016  0:00 /sbin/getty -8 38400 tty4
root     377  0.0  0.0 12792  1876 ?      Ss+  2016  0:00 /sbin/getty -8 38400 tty2
root     378  0.0  0.0 12792  1884 ?      Ss+  2016  0:00 /sbin/getty -8 38400 tty3
root     392  0.0  0.0 23660  2176 ?      Ss   2016  2:41 cron
root     404  0.0  0.0 61372  4848 ?      Ss   2016  0:00 /usr/sbin/sshd -D
mysql    440  0.1  1.4 1339004 129800 ?     Ssl  2016 1076:44 /usr/sbin/mysqld
root     518  0.0  0.0 12792  1900 ?      Ss+  2016  0:00 /sbin/getty -8 38400 console
root     520  0.0  0.0 63136  2644 ?      Ss   2016  0:00 /bin/login --
ubuntu   653  0.0  0.0 21096  3080 ?      S    2016  0:00 -bash
root     665  0.0  0.0 47840  3344 ?      S    2016  0:00 sudo -i
root     666  0.0  0.0 21120  3032 ?      S+   2016  0:00 -bash
root    16020  0.0  0.2 326512 26500 ?     Ss   Mar14  0:31 /usr/sbin/apache2 -k start
www-data 22531  0.3  1.3 370572 125820 ?     S    Mar19  5:55 /usr/sbin/apache2 -k start
www-data 22533  0.3  1.2 357004 112560 ?     S    Mar19  5:45 /usr/sbin/apache2 -k start
```

Mỗi Process được điều khiển bởi 1 bộ điều khiển tiến trình (**Process Control Block – PCB**), là một cấu trúc dữ liệu do OS cấp chứa các thông tin về Process bao gồm: Định danh (**Identification – Id**), bộ đếm chương trình (**Program Counter**), giá trị thanh ghi của CPU, và độ ưu tiên (**Priority**).

Khi CPT có nhiều Process thì bộ định thời (**Scheduler**) sẽ lên lịch cho các Process thay phiên chạy, tức là chọn thứ tự thực hiện các Process sao cho việc sử dụng CPU là 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.

Một số giải thuật định thời:

- **FIFO (First In First Out)**: Process nào đến trước được xử lý trước.
- **SJF (Shortest Job First)**: Process nào có thời gian cần xử lý nhỏ nhất được xử lý trước.
- **SRT (Shortest Remaining Time)**: Process có thời gian cần xử lý còn lại nhỏ nhất được xử lý trước.
- **Round Robin**: Chia thời gian thành những đoạn bằng nhau gọi là quantum time q. Mỗi Process được xử lý trong một khoảng thời gian q, nếu hết thời gian q mà Process chưa kết thúc thì Process sẽ được đưa xuống cuối hàng đợi (**Queue**) để chờ đợi xử lý sau.

Trong quá trình thực thi, các Process có thể tranh chấp tài nguyên (**Resource**) dẫn đến trì hoãn. Do đó, OS cần phải giải quyết tranh chấp bằng:

- Giải thuật phần mềm.
- Phần cứng có hỗ trợ các lệnh đặc biệt.
- Biến “Semaphore” được quản lý bởi OS.

e. Memory Management (Quản lý bộ nhớ)

OS có các chiến lược quản lý RAM:

- **Fetch Strategy** (Chiến lược nạp): Xác định thời điểm lấy dữ liệu hoặc chương trình nạp vào RAM từ HD, nạp theo yêu cầu hoặc có dự đoán trước.

- **Placement Strategy** (Chiến lược đặt): Gồm 3 giải thuật là chọn vùng nhớ trống đầu tiên - **First Fit**, chọn vùng nhớ trống vừa khít - **Best Fit** và chọn vùng nhớ trống lớn nhất - **Worst Fit**.
- **Allocation Strategy** (Chiến lược cấp phát): Cấp phát liên tục hoặc rời rạc.

Trên OS Windows, mở “Task Manager”/tab “Performance” để xem tình hình RAM.

Trên OS Linux, chạy lệnh “**top -c**” để xem tình hình RAM.

f. Disk Management (Quản lý đĩa cứng)

Mỗi truy cập trên HD của OS gồm 3 thao tác:

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

OS định thời truy cập HD bằng các giải thuật sau:

- **FCFS (First Come First Served)**: Truy cập đến trước được phục vụ trước.
- **SSTF (Shortest Seek Time First)**: 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.

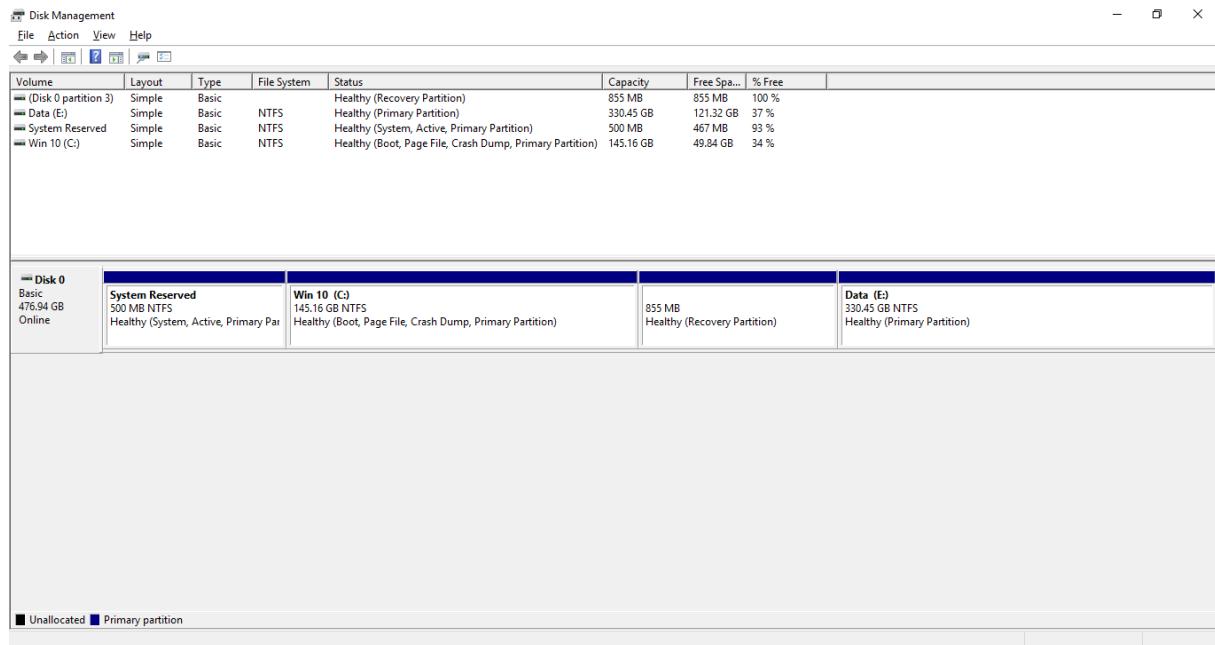
Vì dung lượng RAM có hạn mà số lượng Process được thực hiện rất nhiều (như User vừa nghe nhạc vừa gõ văn bản vừa chạy một mớ các chương trình khác) nên OS tạo ra bộ nhớ ảo (**Virtual Memory**) sử dụng một phần HD để giải quyết vấn đề hết RAM.

Đây là kĩ thuật tách rời địa chỉ mà Process truy cập và địa chỉ của RAM. Địa chỉ mà Process truy cập được ánh xạ từ HD đóng vai trò là Virtual Memory lên RAM khi cần.

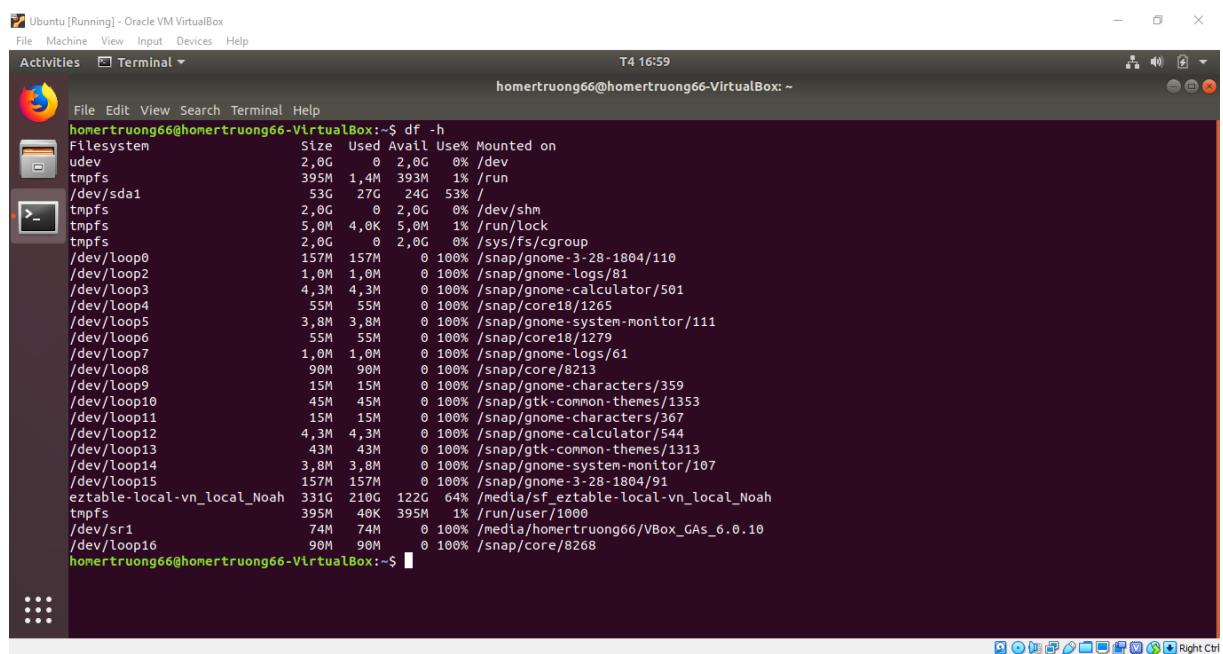
Phép ánh xạ sẽ thực hiện trên từng khối bộ nhớ (**Memory Block**) với 3 phương pháp:

- **Paging** (Phân trang): Chia Virtual Memory 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 (Resident Bit=1) hay chưa (Resident Bit=0: Default), một địa chỉ của khối trên HD và một địa chỉ của khối trên RAM sẽ được ánh xạ lên.
- **Segmentation** (Phân đoạn): Chia Virtual Memory thành các khối kích thước khác nhau.
- **Paging & Segmentation**: Phân đoạn trước rồi phân trang trong mỗi đoạn.

Xem tình trạng HD trên OS Windows:



Xem tình trạng HD trên OS Linux:



g. I/O Management (Quản lý nhập xuất)

Mỗi Process đang chạy thường gặp việc truy cập IO theo 2 cơ chế sau:

- **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 của OS:

- **Buffering:** dành RAM lưu trữ data tạm thời cho việc truyền data.
- **Caching:** Write Back (Flush) và Write Through.
- **Spooling** (chiến lược cho Printer): Lưu trữ đầu ra của Process và định thời xuất ra thiết bị sau đó.

h. Self-Experience

- 1) Trình bày các OS phổ biến
- 2) Nếu bạn đang sử dụng OS Windows thì hãy cài thêm OS Linux (Ubuntu distribution) theo các bước sau:
 - a. Cài Virtual Box (<https://www.virtualbox.org/wiki/Downloads>)
 - b. Tải bản Ubuntu dạng File .iso (<http://releases.ubuntu.com/>)
 - c. Chạy Virtual Box, tạo mới 1 máy ảo (**Virtual Machine**) với loại là Linux
 - d. Cài Ubuntu lên máy ảo vừa tạo sử dụng File .iso
 - e. Chạy thử Ubuntu trên máy ảo
- 3) Xem tình trạng CPU trên OS Windows và OS Linux
- 4) Xem tình trạng Process trên OS Windows và OS Linux
- 5) Xem tình trạng RAM trên OS Windows và OS Linux
- 6) Xem tình trạng HD trên OS Windows và OS Linux

3) Software – SW (Phần mềm)

Đến đây chúng ta đã tìm hiểu mảng CST, bao gồm HW của CPT. Chỉ với HW & OS thuận túy thì User chỉ tương tác với CPT qua giao diện dòng lệnh (**Command Line Interface – CLI**) rất là hạn chế về các tính năng mà User mong muốn.

Do đó phần mềm ứng dụng (**Application Software**) với giao diện đồ họa (**Graphic UI**) ra đời cung cấp nhiều tiện ích cho User. Ghi chú nhỏ là ngoài Application Software còn có phần mềm hệ thống (**System Software**) phục vụ cho việc vận hành hệ thống mà.

Một số SW phổ biến cung cấp các tiện ích cho User:

- SW **Win Word** giúp User soạn văn bản.
- SW **Media Player** giúp User nghe nhạc.
- SW **Photoshop** giúp User thiết kế hình ảnh.
- SW **Chrome** giúp User lướt các trang web.
- SW **Skype** giúp các User nói chuyện với nhau qua mạng.



a. Closed-Source Software – CSSW (Phần mềm nguồn đóng)

Phần mềm nguồn đóng là những SW mà mã nguồn (**Source Code**) của công ty/cá nhân viết ra không được công khai cho cộng đồng IT biết.

Các CSSW thường chạy trên nền OS Windows, là một OS nguồn đóng. Ví dụ: Microsoft Word, Microsoft Excel, Adobe Photoshop...

b. Open-Source Software – OSSW (Phần mềm nguồn mở)

Phần mềm nguồn mở là những SW được xây dựng, sử dụng, phân phối, sửa đổi theo những nguyên tắc nguồn mở (**Open-Source Rule**):

- Qui định quyền sao chép SW và phân phối các bản sao chép.
- Qui định quyền truy nhập Source Code để có thể cải tiến SW theo ý mình, và những phần cải tiến này phải mở ra để cộng đồng nguồn mở biết những sự thay đổi.

Các OSSW thường chạy trên OS Linux, là một OS nguồn mở.

Linux được phát triển từ hệ **UNIX** (OS chủ yếu dành cho các máy chủ), 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.

Không giống như OS Windows chỉ do một mình công ty Microsoft sản xuất, OS Linux do nhiều công ty tạo ra nên rất đa dạng. Các nhà phát triển OS Linux đang cố gắng xây dựng OS Linux ngày càng gần gũi với User giống như OS Windows.

Tuy nhiên, cái chung nhất là tạo ra các OSSW chạy trên OS Linux càng thân thiện và càng tương thích với OS 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 của OSSW:

Ưu điểm	Nhược điểm
<ol style="list-style-type: none"> 1. Độc lập: không bị lệ thuộc nhà cung cấp nào 2. Bảo đảm an toàn và riêng tư: không có “hộp đen” (Black Box) và “cửa hậu” (Back Door) 3. Tích thích ứng cao: sửa đổi dễ dàng để thích ứng với yêu cầu mới 4. Tuân thủ các chuẩn 5. Không bị hạn chế về sử dụng 6. Tính lâu dài: được phát triển bởi “everyone” 7. Phát triển thêm dễ dàng 	<ol style="list-style-type: none"> 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 OSSW. 2. Không có bảo đảm rằng một phát triển cụ thể nào đó sẽ thực hiện được. 3. Năng lực hạn chế của User, nhất là trong các cơ quan công quyền.

c. Software as a Service – SaaS (Phần mềm là dịch vụ)

Một dạng CSSW rất phổ biến những năm gần đây là SaaS.

Với SaaS, User không cần phải bỏ chi phí lớn từ đầu để mua nguyên 1 phần mềm nhiều khi xài không hết tính năng hoặc có nhiều thời điểm không cần xài, thay vào đó User dùng các gói thuê bao (**Subscription Package**) để xài SW khi có nhu cầu, trả tiền theo tuần/tháng/năm hoặc tính năng/lượng dữ liệu tùy loại gói thuê bao.

Ví dụ một số SaaS phổ biến:

- **Salesforce CRM** dùng để quản lý quan hệ khách hàng.
- **Google Docs, Dropbox** dùng để lưu trữ dữ liệu trên mạng.
- **Zoom** dùng để họp trực tuyến.

d. Self-Experience

1. Trình bày các khái niệm CSSW, OSSW, SaaS.
2. Liệt kê những CSSW mà bạn đã/đang/sẽ sử dụng.
3. Liệt kê những OSSW mà bạn đã/đang/sẽ sử dụng.
4. Liệt kê những SaaS mà bạn đã/đang/sẽ sử dụng.

4) Database – DB (Cơ sở dữ liệu)

SW là 1 chương trình lúc chạy được nạp lên RAM dưới dạng một hoặc nhiều Process. Trong quá trình User tương tác với SW thì dữ liệu do User tạo ra như sản phẩm (Product), đơn hàng (Order) cũng sẽ được lưu trữ tạm thời trên RAM. Khi tắt SW, tất cả dữ liệu của SW đang chạy ở trên RAM sẽ mất do giải phóng bộ nhớ cho các SW khác xài. Vì vậy cần phải có chỗ để lưu dữ liệu của SW lâu dài để User có thể tương tác sau khi SW tắt và chạy lại.

Giải pháp dễ thấy nhất để lưu dữ liệu của SW lâu dài là lưu dữ liệu vào File khi User tương tác với SW. Sau đó SW có bị tắt đi thì khi mở lại sẽ đọc dữ liệu từ File lên cho User tương tác tiếp. Giải pháp này chỉ hiệu quả cho các SW đơn giản.

Với những SW có yêu cầu phức tạp hơn, ví dụ: xuất ra 1 báo cáo bán hàng trong quý 3,4/2019 từ những khách hàng thuộc khu vực TPHCM có đơn hàng trị giá ít nhất 10 triệu VNĐ trở lên, thì việc lưu dữ liệu vào File sẽ không xử lý được dữ liệu.

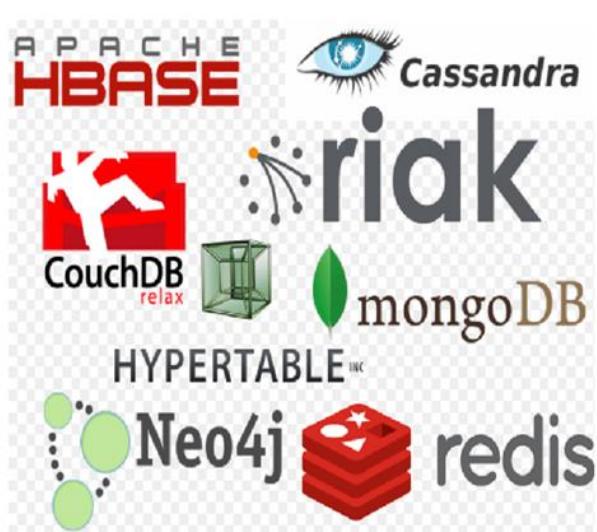
Lúc này chúng ta cần 1 giải pháp khác tốt hơn, đó là cơ sở dữ liệu (**Database – DB**).

DB là một hệ thống lưu dữ liệu có tổ chức để truy xuất/cập nhật dữ liệu tiện lợi, nhanh chóng.

Relational Database

vs

NoSQL Database



Có 2 loại DB: DB quan hệ (Relational DB) & DB NoSQL (NoSQL DB).

Chúng ta sẽ xem xét từng loại ngay dưới đây.

a. Relational Database (Cơ sở dữ liệu quan hệ)

Cơ sở dữ liệu quan hệ là hệ thống lưu trữ dữ liệu trong các bảng (**DB Table**) và các bảng có các quan hệ (**Table Relationship**) với nhau. Trong mỗi Table sẽ có nhiều cột (**Table Column**) thể hiện các thuộc tính của Table. Dữ liệu sẽ được lưu thành từng dòng (**Table Row**). Ví dụ:

- Table: “**orders**”
 - Columns: “**id**”, “**title**”, “**total**”, “**created_at**”
 - Rows:
 1. "dfa35t", "O1", 15000000, "02-02-2020"
 2. "hg4ghsd", "O2", 36000000, "02-02-2020"
- Table: “**order_lines**” quan hệ N:1 với “**orders**” (1 Order có nhiều Order Lines)
 - Columns: “**id**”, “**product**”, “**price**”, “**quantity**”, “**order_id**” (*FK*)
 - Rows:
 1. "xdsg2", "iPhone6", 9000000, 1, "dfa35t"
 2. "x3knad", "Speaker", 3000000, 2, "dfa35t"
 3. "uyt36lhh", "MacBookPro", 36000000, 1, "hg4ghsd"

Dữ liệu được tạo/cập nhật/truy xuất/xóa dùng ngôn ngữ truy vấn cấu trúc (**Structured Query Language – SQL**). Bạn có thể học SQL qua cuốn sách “SQL for MySQL Developers 2e_Rick” hoặc học bằng các giáo trình tự học (**Tutorial**) trên Internet như <https://www.w3schools.com/sql> and <https://www.tutorialspoint.com/sql/index.htm>.

Một số DB quan hệ phổ biến: **MySQL**, **Microsoft SQL Server**, **Oracle DB**, **PostgreSQL**...

b. Entity Relationship Diagram – ERD (Mô hình quan hệ thực thể)

Mỗi Table trong DB quan hệ thể hiện 1 lớp đối tượng ngoài đời gọi là thực thể (**Entity**). Ở ví dụ trên thì bảng “**orders**” thể hiện thực thể **Order** (đơn hàng) ngoài đời.

Mỗi DB quan hệ sẽ có rất nhiều Table có quan hệ chồng chéo với nhau. Nếu chỉ nhìn vào danh sách các Table của 1 DB quan hệ thì chúng ta sẽ không thể hiểu được cái DB này đang chứa những dữ liệu gì, chúng quan hệ ra sao.

Do đó, mô hình quan hệ thực thể (**Entity Relationship Diagram – ERD**) ra đời đem đến góc nhìn tổng quát về cấu trúc của DB quan hệ.

Xem ví dụ: https://en.wikipedia.org/wiki/Entity%20relationship_model

c. Database (De)Normalization ((Phá) Chuẩn hóa DB)

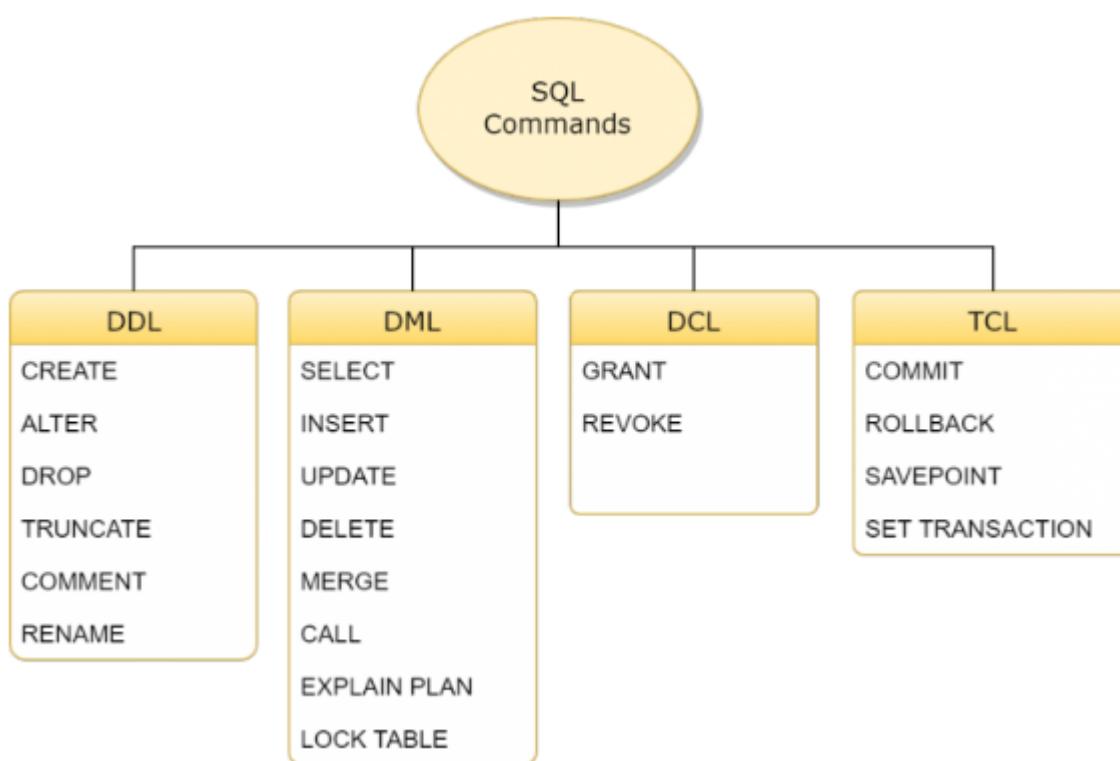
Chuẩn hóa DB (**Database Normalization**) là việc thiết kế DB theo các dạng chuẩn DB (**Database Norm**) để đảm bảo DB có tính toàn vẹn dữ liệu (**Data Integrity**).

Phá chuẩn hóa DB (**Database Denormalization**) là việc thay đổi một/vài DB Norm nhằm mục đích nào đó như truy xuất dữ liệu nhanh hơn, ví dụ ở trên: lưu thêm 1 Column trong Table “orders” là “total” cho biết tổng giá trị 1 Order để truy xuất nhanh hơn thay vì tính từ 1 danh sách Order Lines (lưu trong Table “order_lines”) của Order.

d. Structured Query Language – SQL (Ngôn ngữ truy vấn cấu trúc)

SQL là công cụ để truy xuất và cập nhật dữ liệu vào DB quan hệ một cách thuận tiện và nhanh chóng.

SQL bao gồm một số thành phần là DDL, DML, DCL và TCL được thể hiện trong hình bên dưới đây:



e. Data Definition Language – DDL (Ngôn ngữ định nghĩa dữ liệu)

Ngôn ngữ định nghĩa dữ liệu (**Data Definition Language – DDL**) là ngôn ngữ cung cấp các lệnh SQL liên quan đến cấu trúc DB:

- **CREATE**: Tạo DB, Table, Index, Function, View, Stored Procedure, Trigger.
- **ALTER**: Thay đổi cấu trúc DB.
- **DROP**: Hủy các đối tượng trong DB.
- **RENAME**: Thay đổi tên của 1 đối tượng trong DB.
- **TRUNCATE**: Xóa toàn bộ dữ liệu của 1 Table.
- **COMMENT**: Thêm bình luận vào tự điển dữ liệu.

f. Data Manipulation Language – DML (Ngôn ngữ xử lý dữ liệu)

Ngôn ngữ xử lý dữ liệu (**Data Manipulation Language – DML**) là ngôn ngữ cung cấp các lệnh SQL để xử lý dữ liệu trong DB:

- **SELECT**: truy vấn dữ liệu, các trường hợp truy vấn dữ liệu phổ biến:
 - Truy vấn trên 1 Table
 - Truy vấn trên nhiều Table dùng phương pháp kết bảng (**JOIN**) hoặc các câu truy vấn con (**Sub Query**)
 - Truy vấn gom nhóm (**GROUP BY**) và dùng các hàm tổng hợp (**Aggregate Function**)
- **INSERT**: nhập dữ liệu vào 1 Table.
- **UPDATE**: thay đổi dữ liệu trong 1 Table.
- **DELETE**: xóa một/nhiều Row trong 1 Table.
- **MERGE - UPSERT**: nhập (nếu chưa có) hoặc thay đổi (nếu đã có) dữ liệu trong 1 Table.
- **CALL**: gọi 1 PL/SQL hay là 1 chương trình con Java
- **EXPLAIN PLAN**: diễn giải 1 đường dẫn truy cập dữ liệu (**Data Access Path**)
- **LOCK TABLE**: khóa 1 Table để tránh truy cập đồng thời (**Concurrent Access**)

g. Data Control Language – DCL (Ngôn ngữ điều khiển truy cập dữ liệu)

Ngôn ngữ điều khiển truy cập dữ liệu (**Data Control Language – DCL**) là ngôn ngữ cấp/hủy quyền truy cập DB cho các chương trình/User như **GRANT**, **REVOKE**.

h. Database Transaction (Giao dịch DB)

Database Transaction là 1 danh sách tuân tự các lệnh SQL đọc dữ liệu hoặc thay đổi dữ liệu trong DB. Để đảm bảo Data Integrity trong DB, DB Transaction có 4 tính chất **ACID**:

- 1) **Atomicity** (đơn nhất): tất cả các lệnh phải thành công hết, nếu 1 trong các lệnh thất bại thì kết quả của các lệnh đã được thực hiện trước đó sẽ bị hủy, cái này gọi là tác vụ được tất hoặc không có gì (**All-or-Nothing Operation**).
- 2) **Consistency** (nhất quán): DB đang ở trạng thái nhất quán (**Consistent State**), sau mỗi DB Transaction thì DB phải được duy trì ở Consistent State.
- 3) **Isolation** (tách biệt): Nếu có nhiều DB Transaction cùng chạy thì chúng phải được tách biệt hoàn toàn, không ảnh hưởng nhau.
- 4) **Duration** (lâu dài): Các kết quả được cập nhật xuống DB phải tồn tại lâu dài bất chấp phần cứng hay phần mềm của DB Server có vấn đề.

Database Commit/Rollback (Thực thi/Hủy trong DB): Là cách ra hiệu cho DB ghi dữ liệu vào các Table hoặc hủy các lệnh SQL đã chạy trước đó để kết thúc 1 DB Transaction.

i. Transaction Control Language – TCL (Ngôn ngữ điều khiển giao dịch)

Ngôn ngữ điều khiển giao dịch (**Transaction Control Language – TCL**) là ngôn ngữ giúp điều khiển thành công/hủy giao dịch như **COMMIT**, **ROLLBACK**.

j. NoSQL Database (Cơ sở dữ liệu NoSQL)

DB NoSQL là hệ thống lưu trữ dữ liệu không dùng các bảng và các quan hệ như DB quan hệ mà dùng mô hình “**Key – Value**” để lưu trữ và truy xuất dữ liệu rất nhanh. Chỉ cần đưa “**Key**” vào là lấy được “**Value**” ngay.

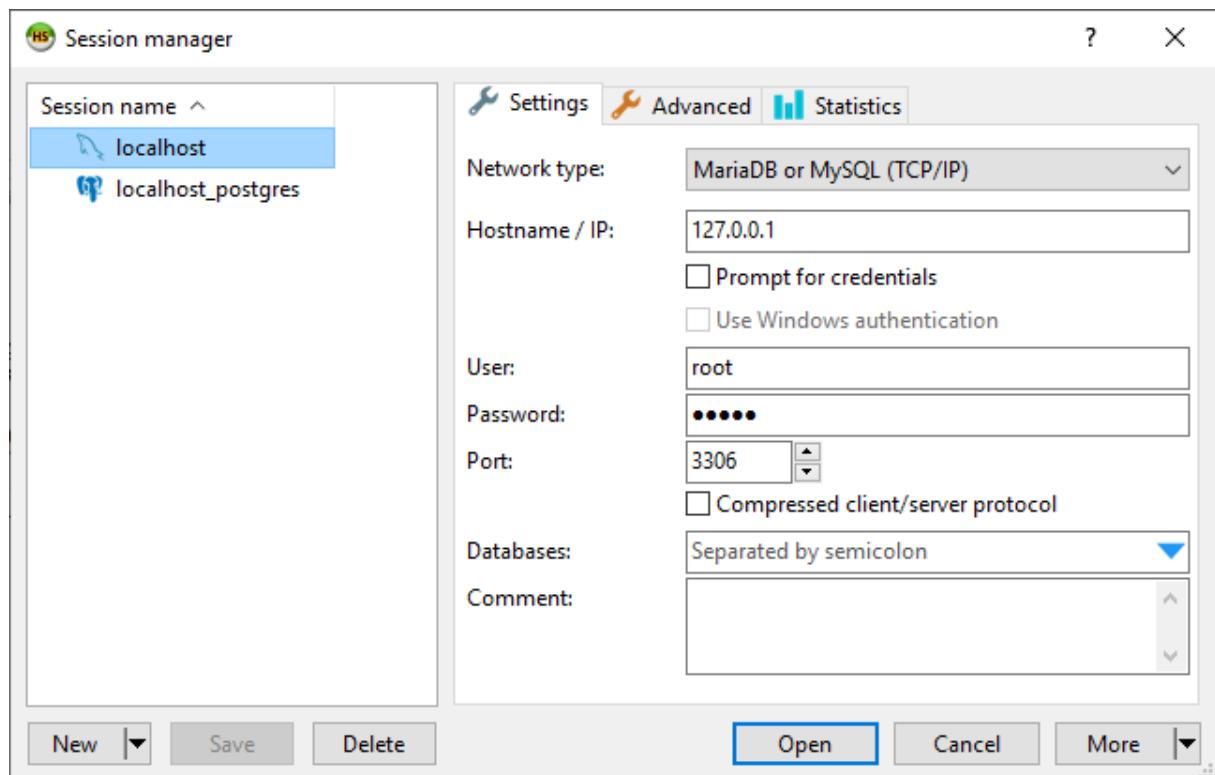
Một số loại DB NoSQL phổ biến: **Cassandra** (được Facebook sử dụng từ thuở đầu), **MongoDB**, **Redis**, **Riak**, **Memcache**...

Ngoài việc dùng “**Key**” để lấy “**Value**”, một số loại DB NoSQL cũng hỗ trợ ngôn ngữ riêng để truy vấn dữ liệu đa dạng hơn, như MongoDB có Document Query Language (<https://docs.mongodb.com/manual/tutorial/query-documents/>).

k. Database Management System – DBMS (Hệ quản trị DB)

Hệ quản trị DB (**Database Management System – DBMS**) là công cụ giúp chúng ta quản trị các DB thuận tiện và dễ dàng.

Ví dụ, HeidiSQL là 1 công cụ giúp quản lý các DBMS hiệu quả và tiện dụng:



I. Database Selection (Lựa chọn DB)

Tùy theo lĩnh vực và yêu cầu của SW mà chúng ta sẽ chọn DB quan hệ hoặc DB NoSQL:

- Nếu mô hình dữ liệu có quan hệ phức tạp, chồng chéo nhau -> chọn DB quan hệ. Ví dụ như dự án “iVision” cung cấp giải pháp khám bệnh từ xa mà tôi đã làm bên Canada sử dụng DB quan hệ là MySQL.
- Nếu mô hình dữ liệu đơn giản, nhiều dữ liệu và đòi hỏi truy xuất nhanh -> chọn DB NoSQL. Ví dụ như dự án “GPM” quản lý sản phẩm xuất nhập khẩu tôi đã làm bên Canada sử dụng DB NoSQL là MongoDB.

(Tham khảo 2 dự án “iVision” & “GPM” trong cuốn hồi ký “Hành trình ĐAM MÊ IT” của tôi).

Khi đã chọn được 1 trong 2 loại DB, tiếp theo chúng ta sẽ phải chọn nhà cung cấp DB (MySQL, MS SQL, Oracle, 10gen...) tùy thuộc vào nhiều yếu tố như yêu cầu của SW, chi phí dự án, tính năng DB cung cấp...

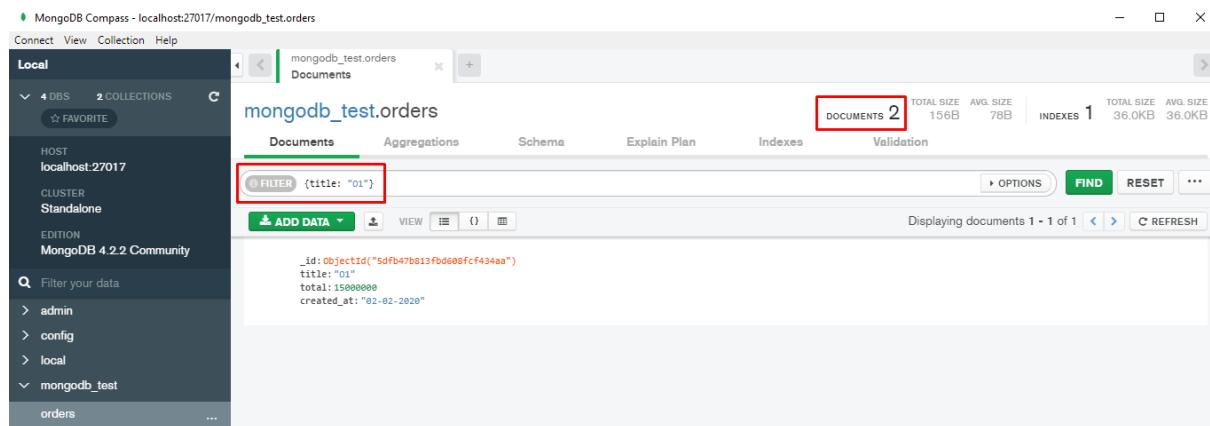
m. Self-Experience

1. Trình bày cơ sở dữ liệu là gì, có mấy loại, khi nào sử dụng loại nào?

2. Trải nghiệm DB quan hệ: MySQL
 - Cài MySQL: <https://www.mysql.com/downloads/>
 - Cài HeidiSQL (công cụ thao tác DBMS): <https://www.heidisql.com/download.php>
 - Chạy HeidiSQL, login vào MySQL local (127.0.0.1:3306).
 - Tạo 1 DB tên là “mysql_test”.
 - Tạo 2 Table “orders” và “order_lines” với các Column & Row như ví dụ ở trên.
 - Chuyển sang tab “Query” trên HeidiSQL, thực hành các câu lệnh **INSERT, SELECT, UPDATE, DELETE** trên 2 Table “orders” và “order_lines”.

3. Trải nghiệm DB NoSQL: MongoDB
 - Cài MongoDB: <https://www.mongodb.com/download-center/community>
 - Cài MongoDB Compass (công cụ thao tác MongoDB): <https://www.mongodb.com/products/compass>
 - Chạy MongoDB Compass, login vào MongoDB local (127.0.0.1:27017).
 - Tạo 1 DB tên là “mongodb_test”.
 - Tạo 2 tập hợp (Collection) “orders” và “order_lines” với dữ liệu như ví dụ trên.
 - Thực hành các thao tác trên 2 tập hợp “orders” và “order_lines”:

<https://docs.mongodb.com/manual/tutorial/query-documents/>



5) Computer Network – CNE (Mạng máy tính)

Với HW, OS, SW và DB thì User đã có 1 hệ thống IT đủ để thực hiện các chức năng họ mong muốn trên 1 CPT của User.

Tuy nhiên có những SW chạy trên 1 CPT cần truy xuất dữ liệu trên 1 CPT khác như ví dụ chuyển tiền từ VCB qua ACB ở trên thì cần phải có mạng máy tính (**Computer Network – CNE**) để thực hiện việc này.



Hai chương trình trên 2 CPT muốn giao tiếp với nhau thì:

- Hai CPT phải kết nối vật lý (**Physical Connection**) với nhau thông qua dây cáp (**Cable**) gắn ở 2 card giao tiếp mạng (**Network Interface Card – NIC**) hoặc kết nối bằng công nghệ không dây (**Wireless Technology**).
- Hai chương trình nói chuyện luận lý (**Logical Connection**) với nhau dựa trên 1 giao thức (**Protocol**) đã được định nghĩa nào đó. Một chương trình đóng vai trò chủ (**Server**) cung cấp các dịch vụ để một chương trình đóng vai trò khách (**Client**) kết nối tới sử dụng dịch vụ. Dựa trên 2 vai trò này mà giới chuyên gia đã đưa ra 1 mô hình tương tác huyền thoại trên CNE: mô hình Khách-Chủ (**Client-Server Model**).

a. CNE History (Lịch sử 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.
- Năm 1983, APARNET & NFSNET sáp nhập lại thành một mạng duy nhất là APARNET.
- Năm **1991**, mạng **Internet** chính thức ra đời còn APARNET chuyển thành mạng MILLINEt phục vụ cho quân đội Mỹ.

b. CNE Types (Các loại mạng máy tính)

Dựa trên qui mô, CNE có các loại 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.

c. CNE Components (Các thành phần trong mạng máy tính)

Các thành phần quan trọng trong CNE:

- **Phần cứng:**
 - Máy tính (**Computer**)
 - Card giao tiếp mạng (**NIC**)
 - Phương tiện truyền (**Medium**)
 - Bộ khuếch đại (**Hub**)
 - Bộ chuyển mạch (**Switch**)
 - Bộ tìm đường (**Router**)
- **Phần mềm:**
 - Hệ điều hành mạng (**Network Operating System**)
 - Phần mềm mạng (**Network Software**)

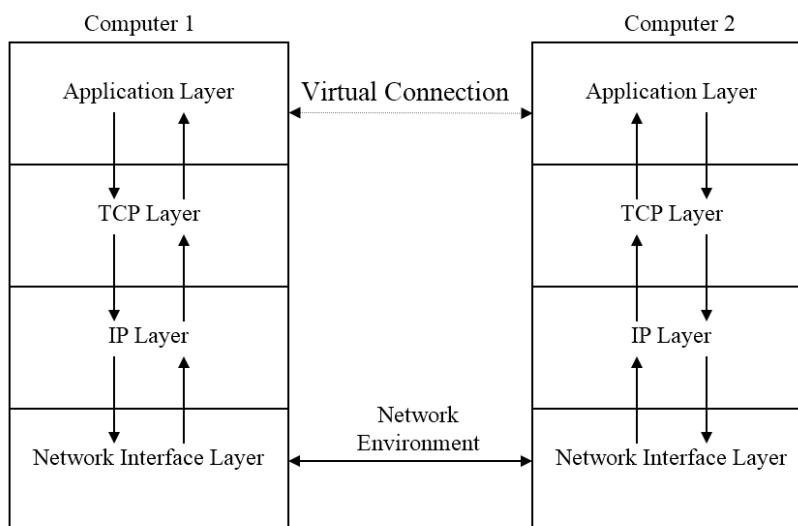
d. OSI Model (Mô hình OSI)

Mô hình lý thuyết của CNE là mô hình **OSI (Open System Interconnect)** có 7 lớp (**Layer**):

1. Lớp vật lý (**Physical Layer**): thực hiện kết nối đường truyền vật lí giữa 2 CPT. Do đó chúng ta cần quan tâm đến 3 vấn đề: Phương tiện truyền, Chuẩn giao tiếp và Thông số đường truyền.
2. Lớp kết nối dữ liệu (**Datalink Layer**): truyền data giữa 2 CPT cho trước và có điều khiển lỗi. Như vậy nó cần phải có 2 module: truyền (**Transferring**) và điều khiển lỗi (**Error Controlling**).
3. Lớp mạng (**Network Layer**): 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**).
4. Lớp vận chuyển (**Transport Layer**): cung cấp giao tiếp giữa 2 ứng dụng trên 2 CPT và đánh địa chỉ tường minh (địa chỉ port). Quản lý kết nối bằng giải thuật bắt tay 3 lần.
5. Lớp phiên làm việc (**Session Layer**): thiết lập, quản lí và kết thúc các phiên làm việc giữa các ứng dụng và dò tìm vị trí các dịch vụ.
6. Lớp trình bày (**Presentation Layer**): mã hóa data, nén data và đảo ngược data.
7. Lớp ứng dụng (**Application Layer**): cung cấp các dịch vụ mạng
 - o Bảo mật trên mạng (**Network Security**): Dùng mật mã hoá (**Cryptography**).
 - o Xác định tên miền dựa vào DNS (**Domain Name Server**)
 - o Quản trị: Dùng SNMP (**Simple Network Management Protocol**)
 - o Cung cấp các dịch vụ phổ biến: **FTP, Email, Web...**

e. TCP/IP Model (Mô hình TCP/IP)

Mô hình CNE thực tế dựa trên 1 mô hình thu gọn của OSI 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 (**Application Layer**) gửi yêu cầu 1 trang Web trên Computer 2 thì yêu cầu đó được chuyển xuống các lớp TCP (**TCP Layer**) & IP (**IP Layer**) để thêm phần Header & Tailer và sau đó được truyền đi bằng các tín hiệu vật lý bởi lớp giao tiếp mạng (**Network Interface Layer**) sang Network Interface Layer của Computer 2 để gửi lên Application Layer của Computer 2.
- Tương tự, trang Web từ Computer 2 sẽ được gửi đến cho Computer 1 theo yêu cầu.
- User chỉ thấy kết nối ảo (**Virtual Connection**) giữa 2 ứng dụng, còn thực tế thì yêu cầu phải đi theo trình tự như trên.

f. IP Address (Địa chỉ IP)

CPT của 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 CPT 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 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 CPT 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 tính kết nối vào Internet thông qua một ISP bằng đường dây cáp. Khi kết nối, ISP sẽ cấp tạm một địa chỉ IP cho máy tính đó. Như vậy, các máy tính không cần phải xin cung cấp một địa chỉ IP riêng lẻ.

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 (0xxxxxxxx : 8bit NetID)
- **Class B:** 128.0.0.0 - 191.255.255.255 (10xxxxxx.. : 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 LAN:

- **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

g. Subnet (Mạng con)

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

h. Domain Name (Tên miền)

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

Dạng của Domain Name: **host.subdomain.domain**

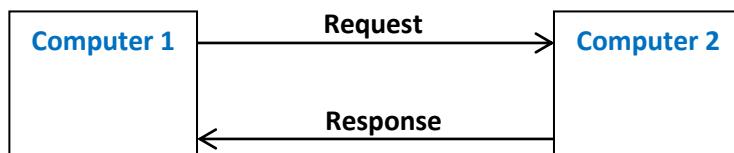
- **host**: là tên máy
- **subdomain**: chỉ ra một tổ chức mạng nhỏ hơn trong domain
- **domain**: định danh cho tên một tổ chức mạng lớn như các công ty, các quốc gia, ...

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

Khi một máy X muốn gửi 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 giúp 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.

i. Protocol (Giao thức)

Giao thức (**Protocol**) là một tập hợp các câu hỏi/đáp giữa 2 CPT giúp chúng nói chuyện với nhau. Protocol được sử dụng trên Internet là giao thức hỏi/đáp (**Request/Response Protocol**).



Protocol định dạng các câu hỏi/đáp và trình tự thực hiện các câu hỏi đáp đó.

j. Networking Application (Ứng dụng có nối mạng)

Xây dựng một ứng dụng có nối mạng (**Networking Application**) sẽ bao gồm các bước:

- Thiết kế 1 chương trình Server
- Thiết kế 1 chương trình Client
- Xây dựng 1 Protocol cho Client & Server giao tiếp với nhau
- Hiện thực chương trình Server
- Hiện thực chương trình Client

Các ứng dụng mạng phổ biến:

- 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**
- Duyệt Web (**Hyper Text Transfer Protocol – HTTP**) : port **80**

k. Web Application (Ứng dụng Web)

Ứng dụng Web (**Web Application**) là một ứng dụng mạng dựa trên mô hình Client-Server nhưng được mở rộng ra thành 3 lớp:

1. **Web Browser** (Trình duyệt Web): gửi yêu cầu lấy về các trang Web từ Web Server.
2. **Web Server**: xử lý các yêu cầu này rồi trả kết quả về cho Web Browser.
3. **Database Server**: nơi lưu trữ dữ liệu để Web Server có thể truy xuất/cập nhật.

Xây dựng một ứng dụng Web là xây dựng các trang Web tương tác với User trên Web Browser và xây dựng các nghiệp vụ phía máy chủ (**Server-Side Logic**) xử lý các tương tác này.

Ví dụ: xây dựng 1 trang Web thương mại điện tử (**eCommerce**) để User chọn hàng, lập đơn hàng... 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 xử lý các tương tác của User.

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

I. Service Port (Cổng dịch vụ)

Do có nhiều loại dịch vụ trên Internet cùng sử dụng chung Protocol, ngoài địa chỉ IP, người ta đưa ra 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 $1 \rightarrow 65535$, trong đó :

- **1 → 999**: các port dành cho các ứng dụng mạng phổ biến như :
 - Port **80** : ứng dụng **Web**
 - Port **25** : ứng dụng **gửi mail SMTP**
 - Port **110** : ứng dụng **trả mail POP3**
- **1000 – 65535**: các Port do chúng ta định nghĩa cho ứng dụng của mình.

m. Popular Internet Services (Các dịch vụ phổ biến trên Internet)

- Dịch vụ tài liệu (**Web**): Dịch vụ này đưa ra cách truy xuất các tài liệu của các máy trên mạng dễ dàng qua các trang Web. Các tài liệu này liên kết với nhau tạo nên các kho tài liệu khổng lồ. Thông qua Internet, các Web Browser như Microsoft Edge, Google Chrome hay Firefox truy cập được thông tin của Web Server bằng **URL (Uniform Resource Locator)** có định dạng: “<http://Host.Subdomain.Domain:Port>”

 - **Host.Subdomain.Domain**: 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 như <http://www.empacbk.com> mà không cần ghi port 80. Khi lập trình Web có xài các Web Server khác thì ta phải thêm phần “:Port” trong URL, ví dụ: <http://localhost:8080>.

- Dịch vụ thư điện tử (**E-mail**): Dịch vụ này cho phép các cá nhân trao đổi thư với nhau thông qua Internet. Để sử dụng dịch vụ này User cần mở một hộp thư (**Mailbox**) tại các máy ISP nơi mình đăng ký dịch vụ Internet hoặc tại các trang Web cung cấp email miễn phí như Gmail, Yandex. Sau khi mở hộp thư, User được cấp một địa chỉ E-mail và một mật mã (**Password**) để đăng nhập vào hộp thư của mình. User 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 định dạng là `mailbox@mailserver.subdomain.domain`.

- Dịch vụ truyền nhận File (**FTP**): Thông qua dịch vụ này Client có thể tải xuống (**Download**) các File từ Server về máy cục bộ hay đưa lên (**Upload**) các File vào Server. Dịch vụ này rất thường được sử dụng để sao chép các phần mềm dạng miễn phí (**Freeware**), chia sẻ (**Shareware**), các bản cập nhật cho chương trình điều khiển thiết bị (**Device Driver**)... Tên của các FTP Server thường có dạng: <ftp.subdomain.domain> (VD : <ftp.empacbk.com>)

n. Self-Experience

1. Trình bày tại sao CNE xuất hiện?
2. Liệt kê các loại CNE theo qui mô.

3. Trong nhà [và công ty] bạn đang xài những loại CNE nào?
4. Trình bày các giao thức phổ biến của CNE.
5. Xây dựng 1 ứng dụng mạng thì cần làm những việc gì.
6. Giải thích ứng dụng Web 3 tầng.
7. Vẽ và giải thích mô hình TCP/IP.
8. Trình bày các dịch vụ phổ biến trên Internet mà bạn biết.

6) Programming (Lập trình)

a. Programming Language – PL (Ngôn ngữ lập trình)

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 CPT thì phải dùng PL trong đó con người viết ra các mã (**Code**) được định nghĩa bởi PL để yêu cầu CPT làm những việc mà con người muốn để tạo thành 1 chương trình máy tính (**Computer Program**). Việc viết Code gọi là lập trình (**Programming**) và người viết Code gọi là lập trình viên (**Programmer**).

b. PL History (Lịch sử ngôn ngữ lập trình)

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

Các PL chỉ khác nhau về cú pháp ngôn ngữ (**Language Syntax**), tức là cách viết Code theo thứ tự nào đó để CPT hiểu, nên việc 1 Programmer chuyển từ PL này sang PL khác trên lý thuyết là không gặp vấn đề gì to tát. Nhưng thực tế thì họ sẽ mất thời gian đáng kể để chuyển đổi PL bởi vì không hẳn bản thân PL mà là các công ty đứng sau lưng PL đó giúp phát triển các thư viện (**Library – Lib**) và các bộ khung (**Framework – FW**) cho việc phát triển phần mềm sử dụng PL đó. Ví dụ như Microsoft đứng sau lưng PL C#, Oracle (trước kia là Sun Microsystems) đứng sau lưng PL Java...

Tuy nhiên, các Lib & FW đều có điểm chung là dựa trên các khái niệm (**Concept**) trong lập trình, vì vậy nếu Programmer nắm vững các khái niệm này thì sẽ rút ngắn được khá nhiều thời gian chuyển đổi PL.

c. Translating Program – TPG (Chương trình dịch)

Khi một PL được định nghĩa ra thì đi kèm với nó sẽ là một chương trình dịch (**Translating Program – TPG**) có chức năng dịch PL này sang Machine Language thì CPT mới thực thi được. Ngoài ra, TPG có thể dịch 1 chương trình được viết bằng 1 PL này sang 1 PL khác.

Có 2 loại TPG:

- **Compiler** (Trình biên dịch): Đây là TPG sẽ dịch toàn bộ các File của Programmer từ đầu đến cuối rồi mới thực thi. Ví dụ: Compiler cho PL Java, C#.
- **Interpreter** (Trình thông dịch): Là TPG vừa dịch vừa thực thi các File của Programmer từng dòng Code cho đến khi kết thúc. Ví dụ: Interpreter cho PL Javascript.

Loại TPG ảnh hưởng đáng kể đến việc lập trình. Một ví dụ cơ bản dễ thấy nhất là việc khai báo biến:

- Đối với PL dùng Compiler: Ta có thể khai báo biến số “x” ở cuối chương trình và xài “x” ở đầu chương trình.
- Đối với PL dùng Interpreter: Ta phải khai báo biến số “x” trước khi xài nó.

Qui trình xây dựng 1 TPG gồm các bước sau:

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

Bất cứ ai có khả năng xây dựng được 1 TPG theo qui trình trên thì hoàn toàn có thể cho ra đời 1 PL mới, đặt tên của họ chẳng hạn. Vấn đề là PL họ tạo ra có đem lại lợi ích nhiều cho cộng đồng Programmer để PL của họ được sử dụng rộng rãi hay không.

d. Structure of a Program (Cấu trúc của 1 chương trình)

Một Program có 1 điểm vào (**Entry Point**) và 1 hoặc nhiều điểm kết thúc (**Exit Point**).

Ví dụ:

- Trong PL Pascal, một Program có cấu trúc:

Begin // Entry Point

...

End // Exit Point

- Trong PL Java, một Program có cấu trúc:

public static void main (String[] args) { // Entry Point

...

if(...){

...

return; // Exit Point (điểm kết thúc có điều kiện)

}

} // Exit Point (điểm kết thúc mặc định khi chạy hết Code)

e. Programming Concepts (Các khái niệm trong lập trình)

- Variable (Biến):** Giống như trong Toán học thì biến là 1 phần tử được định danh có giá trị thay đổi, ví dụ: “age” là 1 biến cho biết tuổi của ai đó, giá trị của “age” đi từ 0->100+.
- Constant (Hằng):** Là 1 biến có giá trị không bao giờ thay đổi, ví dụ: “Pi” luôn là 3.14, đây là 1 hằng do nhà Toán học William Jones định nghĩa, ngoài ra có thể đặt hằng tùy ý như: **const myCountry = “Việt Nam”;** có nghĩa hằng “myCountry” luôn luôn có giá trị là “Việt Nam”.
- Data Type (Kiểu dữ liệu):** Là cách để định nghĩa các tính chất của biến, ví dụ: biến thuộc kiểu nguyên thì sẽ có giá trị từ 0-N (N tùy thuộc vào kích thước của kiểu dữ liệu: 1 Byte (8 bits) thì N là 255 ($2^8 - 1$), 2 Bytes (16 bits) thì N là 65535 ($2^{16} - 1$); kiểu luận lý thì “true/false”.

Có các loại Data Type sau đây:

- **Primitive Type** (Kiểu dữ liệu cơ bản): là các kiểu thông dụng như **Character** (kí tự) **Integer** (số nguyên), **Float** (số thực), **String** (chuỗi), **Boolean** (luận lý).
- **Structured Type** (Kiểu dữ liệu cấu trúc): là kiểu dữ liệu được tổ chức theo một cấu trúc nào đó, mỗi phần tử của kiểu này có một kiểu dữ liệu riêng, có thể là kiểu dữ liệu cấu trúc nữa. Ví dụ: kiểu **Array** (dãy), kiểu **Map** (ánh xạ).
- **Pointer Type** (Kiểu dữ liệu con trỏ): đây là kiểu dữ liệu khá đặc biệt, biến thuộc kiểu này không chứa giá trị của biến mà là chứa địa chỉ tham khảo của ô nhớ chứa giá trị. Vì vậy đây là kiểu dữ liệu rất linh hoạt nếu biết cách xài, còn không thì sẽ gặp rất nhiều rắc rối. Không nhiều PL hỗ trợ kiểu này, PL phổ biến hỗ trợ kiểu này là PL C.
- **Operator** (Phép toán/ Toán tử): Là các tính toán tác động trên một hoặc nhiều biến (gọi là toán hạng (**Operand**)) để cho ra kết quả nào đó, ví dụ:
 - **a++** // tăng biến “a” lên 1 đơn vị
 - **!b** // đảo giá trị biến “b”
 - **c = a + 2**
 - **b = b & d** // AND logic
 - ...
- **Expression** (Biểu thức): Là 1 phép tính trả về 1 kết quả đơn, ví dụ: **int result = 1 + a;**
- **Statement** (Phát biểu): Là 1 đơn vị thực thi hoàn chỉnh, có thể chứa Expression, ví dụ: **print("Hello " + programmingLanguage + " World!");** -> Statement này sẽ in ra màn hình 1 câu “Hello Java World!”.
- **Block** (Khối): Là 1 cách để gom các Expression/Statement liên quan với nhau lại giúp cho việc đọc và quản lý Code thuận lợi hơn, ví dụ:


```
{  
    statement1a;  
    statement1b;  
    statement1c;  
}
```

- **Control Flow** (Luồng điều khiển): Là cách thức điều khiển chương trình chạy theo các nhánh khác nhau tùy thuộc vào các điều kiện cụ thể nào đó. Có các loại luồng sau:
 - **Conditional Flow** (Luồng điều kiện): chương trình phân nhánh theo điều kiện.
 - **Iterative Flow** (Luồng lặp): chương trình sẽ chạy đi chạy lại 1 đoạn Code dựa trên điều kiện nào đó.
- **Exception Handling** (Xử lý lỗi): Là việc Program nhận biết, nắm bắt và xử lý các lỗi có khả năng xảy ra khi Code được thực thi. Có 2 loại lỗi:
 - **Checked Exception**: đây là lỗi mà Programmer đoán trước được sẽ viết Code xử lý để Program không bị chết, ví dụ: lỗi chia 1 biến x cho 1 biến y mà y có thể có giá trị 0, hoặc lỗi chuyển 1 chuỗi (“2020/01-01”) sang ngày/tháng mà giá trị của chuỗi đưa vào không đúng định dạng.
 - **Unchecked Exception**: đây là lỗi mà Programmer không đoán trước được nên không thể viết Code để xử lý, thay vào đó thì sẽ ghi thông tin về lỗi ra màn hình mà Program đang chạy (**Console**) hoặc File (cách phổ biến nhất) hay DB để Programmer thực hiện việc điều nghiên (**Troubleshooting**) và sửa lỗi (**Bug Fixing**) sau khi lỗi diễn ra.
- **Data Structure – DST** (Cấu trúc dữ liệu): Là các cấu trúc được sử dụng để lưu dữ liệu khi User tương tác với Program, các cấu trúc dữ liệu phổ biến gồm **List/Set** (Danh sách/Tập hợp), **Queue/Stack** (Hàng đợi/Chồng), **Map** (Ánh xạ), **Tree** (Cây)…

f. Procedure – PROC (Thủ tục)

Thủ tục (**Procedure – PROC**) là 1 loạt các Statement liên tiếp được đặt trong 1 Block để thực hiện 1 việc gì đó. PROC được đặt tên, nhận vào 1 danh sách các thông số (**Parameter**) và có thể trả về kết quả. Ví dụ:

- `writeToFile(String filename, byte[] content) {...}`
- `float equation1(float a, float b) {...}`
- `Date convertToUTC(Date inputDate) {...}`

-> Khái niệm **PROC Signature**: `writeToFile(String filename, byte[] content);`

PROC được gọi bởi Program hoặc các PROC khác, gọi là cuộc gọi thủ tục (**Procedure Call**). Khi 1 PROC gọi chính nó thì được gọi là cuộc gọi đệ quy (**Recursive Call**), ví dụ như PROC tính giai thừa của 1 số nguyên “n”.

g. PROC-related Concepts (Các khái niệm liên quan đến PROC)

- **Procedure Programming** (Lập trình thủ tục): Là mô hình lập trình trong đó Procedure Call được sử dụng.
- **Formal Parameter** (Thông số hình thức): Là các biến được khai báo trong **PROC Signature**, trong ví dụ trên thì “filename” và “content” là 2 Formal Parameter của PROC “writeToFile”.
- **Real Parameter** (Thông số thực): Khi 1 PROC được gọi, các thông số của nó sẽ được truyền các giá trị vào, các biến chứa các giá trị này gọi là thông số thực. Ví dụ: `writeToFile(inputFilename, uploadedData);` -> “`inputFilename`” và “`uploadedData`” là 2 thông số thực.
- **Parameter Transferring** (Truyền thông số): Có 3 cách truyền thông số cho 1 PROC:
 - **Value Transfer** (Truyền tham trị): việc chỉnh sửa Formal Parameter không ảnh hưởng đến Real Parameter.
 - **Reference Transfer** (Truyền tham khảo): việc chỉnh sửa Formal Parameter ảnh hưởng đến Real Parameter.
 - **Pointer Transfer** (Truyền con trỏ): việc chỉnh sửa Formal Parameter không ảnh hưởng đến Real Parameter, nhưng việc chỉnh sửa ô nhớ mà Formal Parameter tham khảo đến sẽ ảnh hưởng đến Real Parameter.
- **Data Scope** (Tầm vực dữ liệu): Là phạm vi giúp xác định sự tồn tại của một tên biến và giá trị của nó trong trường hợp có nhiều Procedure Call gọi lồng nhau (**Nested Call**). Ví dụ có 1 “**program**” và 2 PROC tên là “**proc1**” & “**proc2**”, cả 3 đều khai báo biến cùng tên là “**x**”, File chứa proc2 có khai báo 1 biến “**b**” bên ngoài proc2.

<code>program {</code>	<code>proc1(int a) {</code>	<code>int b = 6;</code>
<code> int x = 1;</code>	<code> int x = 3;</code>	<code>proc2(int b) {</code>
<code> x = proc1(x);</code>	<code> x = proc2(x+a);</code>	<code> int x = 6;</code>
<code> print(x);</code>	<code> print(x);</code>	<code> print(x);</code>
<code>}</code>	<code> return x--;</code>	<code> return x+b;</code>
	<code>}</code>	<code>}</code>

Kết quả in ra màn hình của lệnh `print(x)` trong proc2, proc1 & program là bao nhiêu?

- **Activity Table** (Bảng ghi hoạt động): Là bảng lưu trữ thông tin của các PROC trong lúc PROC đang hoạt động. Khi PROC kết thúc thì bảng này cũng sẽ bị xóa khỏi bộ nhớ.
- **Caller vs Callee** (PROC gọi vs PROC được gọi): Là 2 tên gọi về quan hệ giữa PROC qua khái niệm Procedure Call. **Lưu ý:** Caller đầu tiên luôn là Program.
- **Call Stack** (Chồng các cuộc gọi): Là nơi lưu trữ thứ tự gọi nhau của các PROC & Program, hiển nhiên Program sẽ nằm dưới cùng và các PROC sẽ nằm ở trên. Khi 1 PROC nào đó kết thúc thì nó sẽ được lấy ra khỏi Call Stack từ trên đầu, còn PROC nào được gọi mới sẽ được bô vào trên đầu. Ví dụ: main -> proc1 -> proc2 -> proc3...
Call Stack phục vụ chủ yếu cho việc tìm lỗi (**Debug**) vì nó giúp Programmer dò theo chồng xem coi lỗi có khả năng xảy ra ở khúc gọi nào, ở thời điểm đó dữ liệu ra sao.
- **Input / Output – I/O** (Nhập/Xuất): Là phương pháp hỗ trợ Program tương tác với các Input Device và Output Device thông qua các PROC được viết sẵn của PL.
- **Remote Procedure Call – RPC** (Gọi thủ tục từ xa): Là cách thức để 2 Program chạy trên 2 vùng nhớ khác nhau (cùng 1 CPT hay 2 CPT khác nhau) có thể gọi PROC của nhau để tương tác theo 1 nghiệp vụ nào đó.

h. Programming Model – PMO (Mô hình lập trình)

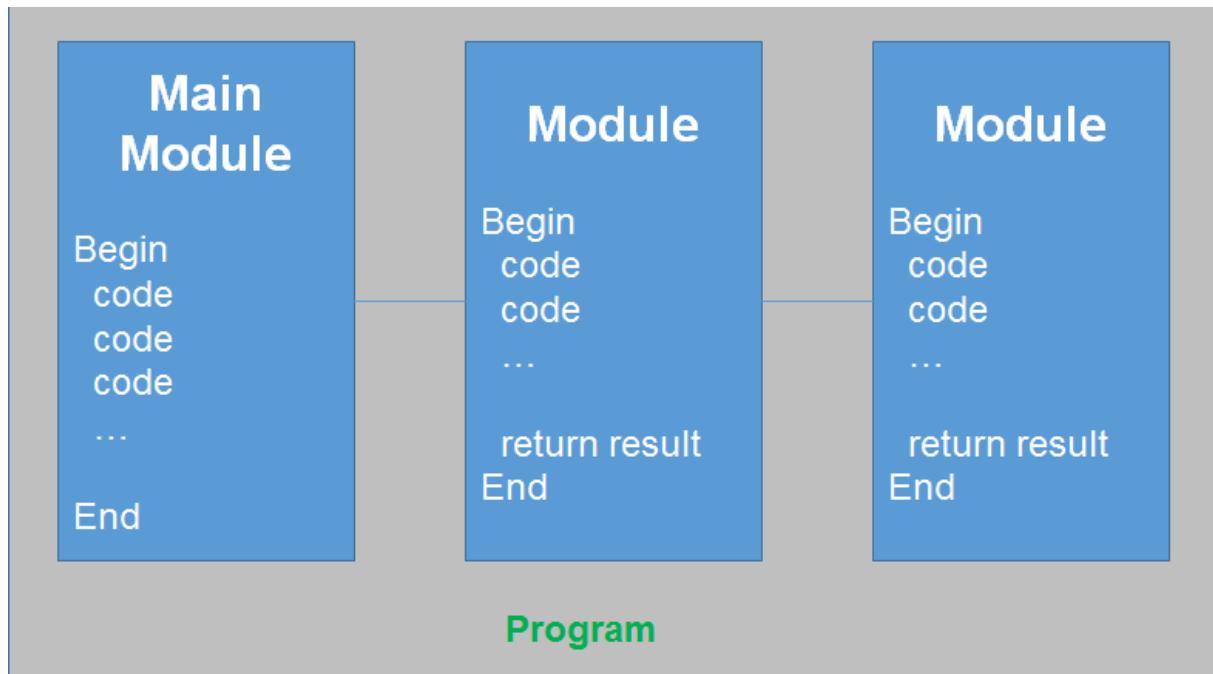
Trong quá trình lập trình, các Programmer có thể sử dụng các mô hình lập trình (**Programming Model – PMO**) sau:

- **Procedure Programming – PP** (PMO thủ tục)
- **Object-Oriented Programming – OOP** (PMO hướng đối tượng)
- **Aspect-Oriented Programming – AOP** (PMO hướng khía cạnh)
- **Functional Programming – FP** (PMO hàm)

Các PMO này không loại trừ lẫn nhau, mỗi PMO giúp Programmer giải quyết 1 vấn đề nào đó, vì vậy chúng có thể tồn tại đồng thời trong 1 Program.

i. Procedure Programming – PP (Lập trình thủ tục)

Trong PP thì Program sẽ có mô đun chính và các mô đun khác. Trong quá trình thực thi thì các đoạn Code trong mô đun chính có thể gọi các PROC của các mô đun phụ khác để làm việc gì đó.



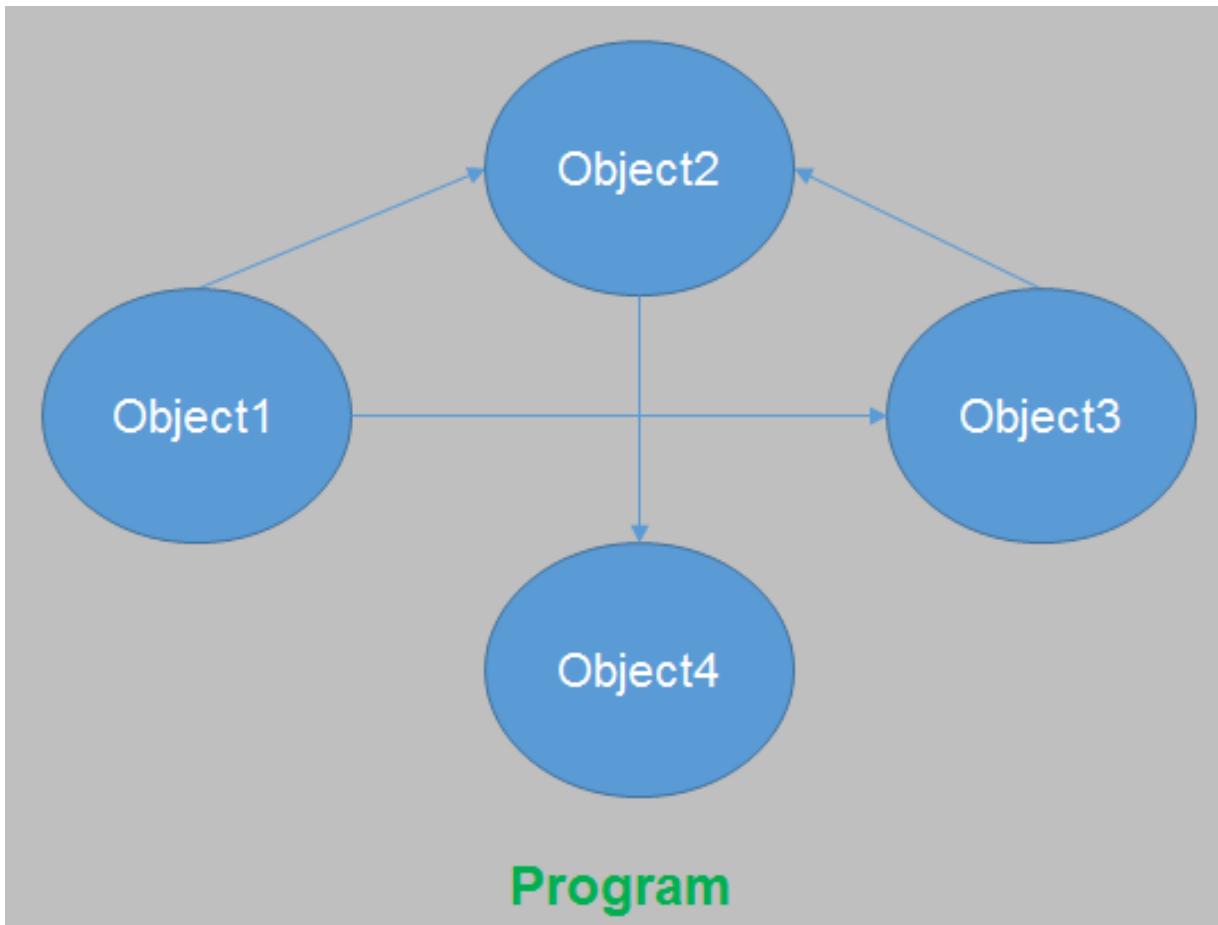
Nhược điểm của PP là:

- Các mô đun **không có khả năng che dấu** các biến và PROC của nó nên bất kỳ mô đun nào cũng có thể truy cập được.
- Các mô đun **không có sự kế thừa** nên khi chúng ta viết một mô đun mới rất giống mô đun đã tồn tại thì phải viết lại hoặc sao chép rồi sửa nội dung làm chúng ta tốn nhiều thời gian và có nhiều Code trùng lặp sẽ 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.

j. Object-Oriented Programming – OOP (Lập trình hướng đối tượng)

Để giải quyết những hạn chế của PP, OOP đã ra đời. Trong OOP, các mô đun được thay bằng các đối tượng (**Object**) thuộc các lớp (**Class**) hay ta gọi là các Object có Type.

Program sẽ bao gồm các Object tương tác với nhau bằng cách gửi thông điệp (**Message**) cho nhau được thể hiện trong hình dưới.



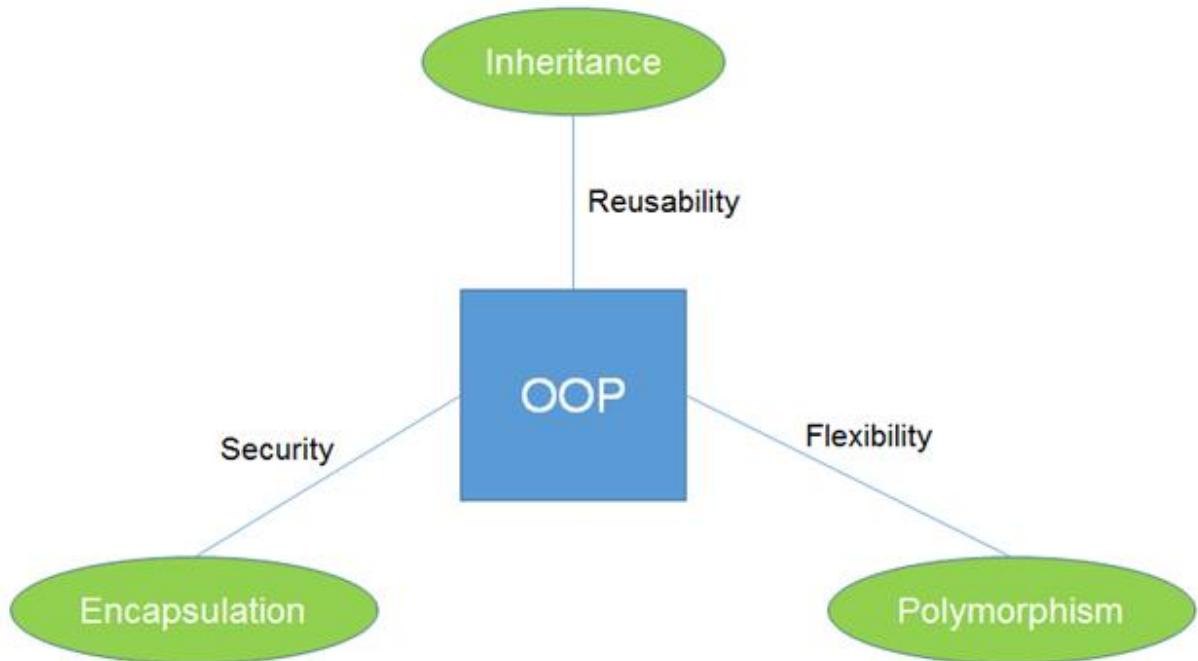
Cách gửi thông điệp của các Object trong OOP thông qua các **Method Call** (cuộc gọi phương thức) tương đương với Procedure Call trong PP.

OOP có 3 thuộc tính cơ bản:

- **Encapsulation** (Tính bao đóng): Object có khả năng che dấu các thuộc tính (**Property**) và phương thức (**Method**) của nó để không cho các Object khác truy cập được. Nó có thể cho các Object khác chỉnh sửa Property của nó bằng các Method nó cung cấp để kiểm soát việc chỉnh sửa này.
- **Inheritance** (Tính thừa kế): Các Object của các lớp con (**Sub Class**) có thể thừa kế các Property và Method của lớp cha (**Super Class**) giúp giảm thời gian và công sức phát triển các Sub Class. Object thuộc Sub Class có thể thay thế Object thuộc Super Class.
- **Polymorphism** (Tính đa hình): Cùng 1 lệnh gọi Method có thể kích hoạt việc thực thi Method của các Object khác nhau, điều này làm cho việc viết Code ngắn gọn và linh hoạt. Ví dụ: Cùng 1 lệnh gọi (*Animal*) *animal.shout()*; có thể kích hoạt Method *shout()* của lớp Cat hoặc Dog tùy lúc khởi tạo biến “animal” từ lớp Cat hay Dog, trong đó lớp Cat & Dog thừa kế lớp Animal. Với tính đa hình này, ta không cần kiểm tra biến “animal” thuộc lớp Cat hay Dog rồi ép về kiểu Cat hay Dog mới gọi Method *shout()*.

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

- **Security** (Tính bảo mật): Các Object chỉ được phép truy cập các Property & Method được cho phép truy cập của các Object khác.
- **Reusability** (Tính tái sử dụng): Các Class đã tồn tại được sử dụng lại để tạo ra những Sub Class nhằm tiết kiệm thời gian và chi phí.
- **Flexibility** (Tính linh hoạt): Sự đa hình giúp viết Code ngắn gọn.



Các khái niệm quan trọng trong OOP:

- **Object** (Đối tượng): Là 1 thực thể (**Entity**) trong “thế giới thực” chứa vấn đề mà Program cần giải quyết” gọi là **Problem Space** (Không gian vấn đề). Entity này sẽ được ánh xạ qua thành 1 Object trong **Solution Space** (Không gian giải pháp) để Program xử lý, ví dụ: 1 Entity là Light (Đèn) trong thực tế được chuyển thành 1 Object là Light trong bộ nhớ chứa Program về quản lý ngôi nhà.
- **Property** (Thuộc tính): Là thành phần cấu tạo nên trạng thái (**State**) của Object, ví dụ: State của 1 Light là: dài 1m, màu đỏ, bật sáng cấp 2 -> Như vậy 3 Property của Light là: **width** (chiều dài), **color** (màu sắc), **level** (mức độ hoạt động).
- **Method** (Phương thức): Là **Behavior** (Hành vi) của Object, ví dụ: Light có các Behavior như bật, tắt, sáng thêm, mờ đi... Các Behavior của Object sẽ thay đổi State của nó.

- **Class** (Lớp): Là tập hợp các Object có cùng các Property và Behavior. Class được đặt tên. Ví dụ: Class Light (Đèn), Dog (Chó), Vehicle (Xe), Order (Đơn hàng), Doctor (Bác sĩ), Patient (Bệnh nhân)... Class được xem là cái khuôn để sinh ra các Object, ví dụ: **Light light1 = new Light();** -> sinh ra 1 Object có tên “light1” từ Class Light.
- **Object Life Cycle** (Vòng đời của Object): Là chu trình từ lúc Object được sinh ra đến khi bị hủy, bao gồm các bước:
 - **Object Instantiation** (Tạo mới Object): Khởi tạo Object từ Class, cấp bộ nhớ cho nó.
 - **Object Construction** (Xây dựng Object): Xây dựng các Property của Object tạo nên State ban đầu (**Initial State**) của Object. Việc này xảy ra trong phương thức khởi tạo (**Construction Method or Constructor**) nằm trong Class.
 - **Object Destruction** (Hủy Object): Giải phóng bộ nhớ mà Object đã chiếm.
- **Object State** (Trạng thái của Object): Là giá trị của các Property của Object ở một thời điểm nào đó.
- **Interface** (Giao diện): Là bộ mặt của Object cho biết các **Service** (Dịch vụ) mà Object cung cấp cho các Object khác tương tác trong Program. Interface được đặt tên. Ví dụ: **interface PersistenceCRUD** cung cấp các dịch vụ **create** (Tạo dữ liệu), **read** (đọc dữ liệu), **update** (cập nhật dữ liệu) và **delete** (xóa dữ liệu) tương tác với DB.
- **Implementation** (Hiện thực): Là 1 Class hiện thực cho 1 Interface, ví dụ:
 - MysqlPersistenceCRUD hiện thực interface PersistenceCRUD trên MySQL DB.
 - MongodtPersistenceCRUD hiện thực interface PersistenceCRUD trên Mongodt DB.
- **Anonymous Class** (Lớp nặc danh): Object được khởi tạo từ Class được đặt tên. Tuy nhiên, Object cũng có thể khởi tạo từ Interface -> trong trường hợp này thì Object này xem như thuộc lớp không có tên. Ví dụ:
 - **PersistenceCRUD obj1 = new MysqlPersistenceCRUD();** -> Ghi chú: obj1 có thể được khai báo kiểu MysqlPersistenceCRUD thay vì kiểu PersistenceCRUD.
 - **PersistenceCRUD obj2 = new PersistenceCRUD() { <phản hiện thực các Service của Interface>};**

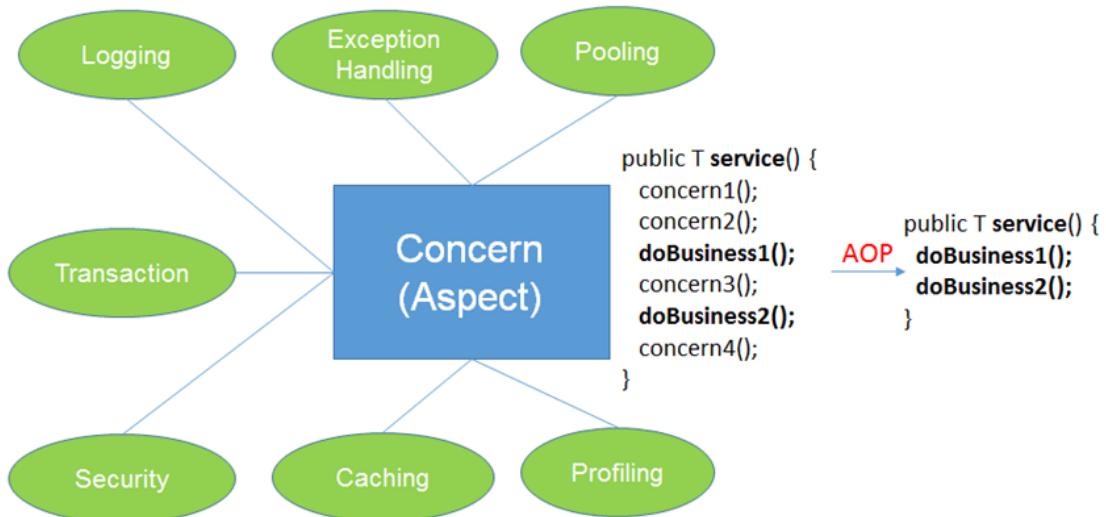
- **Abstraction** (Trừu tượng hóa): Là phương pháp sử dụng những thứ đơn giản để giải quyết các vấn đề phức tạp đằng sau. Abstraction thường được dùng trong phần thiết kế SW sử dụng Abstract Class & Abstract Method. Ví dụ trong 1 Program có dụng đến các hình tròn (**Circle**), hình thoi (**Diamond**), hình ngũ giác (**Pentagon**), lục giác (**Hexagon**)... và tính diện tích (**Area**) của chúng; các loại hình này sẽ được trừu tượng hóa bằng 1 Abstract Class là **Shape** (hình dạng) với 1 Abstract Method là “calculateArea” dùng để tính thể tích, che dấu đi độ phức tạp của việc tính thể tích của từng loại hình.
- **Object Type** (Kiểu đối tượng): Là tính chất để xác định 1 Object thuộc Class nào hoặc Interface nào, ví dụ trên: biến “obj1” thuộc kiểu MysqlPersistenceCRUD (kiểu của Class) hoặc PersistenceCRUD (kiểu của Interface).
- **Variable Scope** (Tầm vực của biến):
 - **Class Variable** (Biến lớp): Là biến được khai báo ở mức Class, được truy cập trực tiếp từ tên Class, ví dụ: MysqlPersistenceCRUD.DB_NAME. Biến lớp thường là các hằng để được sử dụng lại ở nhiều nơi khác ngoài Class nơi nó được khai báo.
 - **Instance Variable** (Biến thực thể): Là Property của Object, 2 từ Object & Instance tương đương nhau.
 - **Local Variable** (Biến cục bộ): Là biến được khai báo trong Constructor, Method hoặc Block.
- **Method Scope** (Tầm vực của Method):
 - **Class Method** (Phương thức lớp): Là Method được khai báo ở mức Class, được truy cập trực tiếp từ tên Class, ví dụ: **Math.sqrt(x)** -> Class Method “sqrt” của Class “Math” dùng để lấy căn bậc hai của 1 biến “x”.
 - **Instance Method** (Phương thức thực thể): Là Method của Object.
 - **Local Method** (Phương thức cục bộ): Là Method của Object mà chỉ được sử dụng bên trong Object, các Object khác không truy cập được.
- **Access Control** (Điều khiển truy cập): Là phương pháp kiểm soát việc truy cập các Property & Method của Object. Thông thường các PL sẽ cung cấp các từ khóa (**Keyword**) như **private** (chỉ truy cập bên trong Object), **protected** (truy cập bên trong Object và Super Class), **public** (mọi Object đều truy cập được). Như vậy với trường hợp Local Method ở trên, từ khóa được xài sẽ là **private** để khai báo các Local Method.

- **Method Overriding** (Phương pháp đè phương thức): Là cách mở rộng/thay thế tính năng của Method thuộc Sub Class so với Method của Super Class.
- **Method Overloading** (Phương pháp quá tải phương thức): Là cách mở rộng tính năng của Method với các danh sách thông số khác nhau, ví dụ: Method *turnOn()* -> bật đèn sáng lên mà không có định giờ, *turnOn(int minutes)* -> bật đèn sáng trong một số phút rồi tự tắt.
- **Generics** (Tính tổng quát hóa): Là phương pháp hỗ trợ tìm ra lỗi về kiểu của Object lúc lập trình thay vì lúc Program chạy, ví dụ: *List<Light> lights;* -> danh sách này chỉ chứa các Object có kiểu Light, nếu đưa Object kiểu khác vào danh sách thì TPG sẽ báo lỗi ngay lúc biên dịch Program chứ không đợi đến khi Program chạy mới ra lỗi.

k. Aspect-Oriented Programming – AOP (Lập trình hướng khía cạnh)

Ngoài OOP, Programmer cần nắm mô hình lập trình hướng khía cạnh (**Aspect-Oriented Programming – AOP**) để giúp cho việc viết Code ngắn gọn để dễ quản lý và bảo trì cũng như phát triển thêm các tính năng khác sau này hơn.

Xét ví dụ trong hình sau:



Cái Method “service” thực thi 2 nghiệp vụ chính là “doBusiness1” và “doBusiness2”.

Tuy nhiên xen giữa việc thực thi 2 nghiệp vụ chính này là các Code để xử lý các khía cạnh (**Aspect or Concern**) khác mà Programmer cần quan tâm như lưu vết hệ thống (**System Logging**), giao dịch nghiệp vụ (**Business Transaction**), phân quyền người dùng (**User Authorization**)... làm cho Method “service” trở nên cồng kềnh không đáng có.

Sau khi ứng dụng AOP, Method “service” chỉ còn tập trung vô đúng 2 nghiệp vụ chính. Tất cả 4 khía cạnh khác được AOP xử lý êm xui trước khi vào Method “service”.

I. Functional Programming – FP (Lập trình hàm)

Ngoài PP, OOP và AOP, Programmer cũng cần nắm vững lập trình hàm (**Functional Programming – FP**) có vai trò rất quan trọng, đại diện tiêu biểu là PL Javascript, trong PL Java có tính năng Lambda Expression, còn trong PL .Net thì có tính năng LINQ.

Trong FP, Function (tương đương khái niệm PROC) được truyền từ Function này sang Function khác như là dữ liệu giúp cho việc lập trình rất linh hoạt.

Xét 1 ví dụ được viết bằng mã giả (Pseudo-Code) sau:

```
Function handleProduct(product) {
    show Product Edit UI;
    bind product data into Product Edit UI;
    hide loading image;
}

Function getProduct(productId, processProduct) {
    show loading image;
    AJAX.loadProduct(productId, (product) -> {
        processProduct(product);
    });
}

// get Product whose id = “Pxyz123” to edit
getProduct(“Pxyz123”, handleProduct);
```

Trong ví dụ này, Function tên là “handleProduct” được truyền vào Function “getProduct” như là dữ liệu, để sau đó nó được gọi để xử lý Product ngay sau khi quá trình lấy Product, là một quá trình xử lý bất đồng bộ (**Asynchronous Processing**), từ Backend hoàn tất. Với việc truyền Function như vậy, luồng điều khiển được chuyển từ Program sang Function “getProduct”, nếu không thì Program sẽ không biết khi nào quá trình lấy Product xong để mà xử lý.

m. Data Structure – DST & Algorithms (Cấu trúc dữ liệu & Giải thuật)

Trong quá trình lập trình, Programmer thường sẽ sử dụng các giải thuật trên các cấu trúc dữ liệu (**Data Structure – DST**) để giải quyết các vấn đề nào đó.

Ví dụ: Programmer 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 Programmer 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 DST có thể sử dụng là List.

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

Các loại DST thông dụng mà tất cả các PL đều hỗ trợ:

- **List** (Danh sách): các phần tử được lưu tuần tự, truy xuất theo chỉ số (**Index**).
- **Set** (Tập hợp): các phần tử được lưu tuần tự và không cho tồn tại 2 phần tử trùng nhau.
- **Tree** (Cây): các phần tử được lưu theo hình cái cây, bắt đầu bằng nút gốc (**Root Node**) và đi xuống các nút trung gian (**Interior Node**) đến tận cùng là nút lá (**Leaf Node**).
- **Map** (Ánh xạ): các phần tử được lưu tuần tự như List, 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 Object nào đó, và cũng có thể là 1 List, 1 Set, 1 Tree, hoặc thậm chí 1 Map.

Ví dụ: chúng ta lưu 1 List các mã số lớp, từ mỗi mã số lớp sẽ lấy ra 1 List sinh viên:

```
.[BKIT98-01 | [Hoàng, Hùng, Minh, Nga]]  
. [BKIT98-05 | [Phương, Thảo, Tiên]]
```

Trên mỗi loại DST, PL sẽ cung cấp sẵn một số lượng các giải thuật cơ bản để Programmer tiện xài mà không cần phải tự suy nghĩ viết ra.

PL nào khác sẽ cung cấp thêm các giải thuật nâng cao khác. PL nào càng mạnh thì số lượng giải thuật càng nhiều cho Programmer.

Nếu Programmer đang xài PL nào chưa hỗ trợ giải thuật nào đó họ cần thì họ có thể đi tìm các Lib hay FW mà những Programmer khác đã viết trước đó (chia sẻ cho cộng đồng hoặc bán).

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

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

Khi chuyển sang PL mới thì Programmer cần làm lại 2 mục sau cùng.

n. Design Patterns (Các mẫu thiết kế)

Dưới đây là các Design Pattern 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

o. Library – Lib (Thư viện) & Framework – FW (Bộ khung)

Lib là nơi chứa các tính năng phổ biến (như giải phương trình bậc 2, tính căn bậc 2...), có thể sử dụng lại, được viết dưới dạng các PROC để Programmer sử dụng cho nhanh (khỏi phải suy nghĩ viết lại tính năng đó) và gọn (Code không bị trùng nhiều chỗ có xài tính năng đó).

FW cung cấp nền móng và các thành phần được xây dựng sẵn giúp cho việc xây dựng 1 Program nhanh hơn.

Lib & FW có mục đích giống nhau nhưng FW có qui mô lớn hơn Lib về cấu trúc, độ phức tạp và luồng điều khiển (FW thường sử dụng khái niệm đảo ngược điều khiển (**Inversion of Control – IoC**)).

p. Debugging (Tìm lỗi)

Sau khi lập trình xong, Programmer hò hỏi chạy thì kết quả không đúng như mong đợi thì Programmer phải làm sao đây?

Chuyện này là chuyện thường ngày ở huyên mà thôi. Programmer phải tìm cách gì đó để đảm bảo những phần Code đã viết chạy đúng ý họ, cái này gọi là “**debug**”. Có 2 cách để debug:

1. Chạy từng dòng Code trong môi trường phát triển tích hợp (**Integrated Development Environment – IDE**).
2. In những thông tin Programmer muôn xem ra Console hoặc File.

Cách số 1 chúng ta có thể làm trong lúc lập trình, nhưng khi chương trình đã đưa đến User 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 SW nhanh hơn và tốt hơn.

q. Code Quality (Chất lượng mã)

Để 1 dự án SW 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 thành viên 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, viết tắt là **BRBS**.

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

UserDao

```
public Id create (User) {  
    beginTransaction();  
    // insert User  
    endTransaction();  
}  
  
public User read (id);  
public Id update (User);  
public void delete (Id);
```

OrderDao

```
public Id create (Order) {  
    beginTransaction();  
    // insert Order  
    endTransaction();  
}  
  
public Order read (id);  
public Id update (Order);  
public void delete (Id);
```

Ta thấy 4 Method create/read/update/delete của 2 Object (User, Order) rất giống nhau mà phải tạo 2 File và định nghĩa đến 8 Method. Trong trường hợp này ta dùng “Generics” thì chỉ cần 1 File đặt tên là “GenericDao” định nghĩa ra 4 Method cho tất cả các loại Object như User, Product, Order, Shipment...

Redundant Code: Điều này nói đến việc các đoạn Code được viết ra nhưng sau đó vì lý do gì đó không được xài chúng nhưng cũng không bị xóa đi.

Một trường hợp khác nữa là các đoạn Code được xài trong phiên bản (**Version**) X, sau đó qua Version X+1 chỉnh sửa một số chỗ dẫn đến các đoạn Code này không còn được xài nữa mà người viết ra nó không nhớ hoặc không hay biết.

Ví dụ: Trong IDE, Code không được sử dụng sẽ có màu xám đen như Method ‘doSomethingComplex()’, khi rò chuột vào, nó sẽ hiển thị thông báo.

```

941 @
942
943
944
945
946
947
948

```

```

private String doSomethingComplex() {
    String result = "init value";
    // biz logic here
    return result;
}

```

Giải pháp cho vấn đề này:

- Xóa Method này. Lưu ý: Method 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 nên xóa nó vì nó có thể được xài từ các Object khác bên ngoài sau này.

HOẶC

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

```

941
942
943
944
945
946
947
948

```

```

// TODO: use this method later
private String doSomethingComplex() {
    String result = "init value";
    // biz logic here
    return result;
}

```

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:

User -> UserRoles -> Role

SecurityService

```

public void insertToken(Id userId, String token) throws Exception {
    User user = orm.read(userId);
    user.setToken(token);
    orm.save(user);
}

```

Ở đây xài công nghệ ORM để đọc nguyên 1 Object User (có thể có rất nhiều Property) từ DB lên, gán giá trị cho 1 Property là ‘token’ rồi lại lưu nguyên 1 Object User xuống DB.

-> Giải pháp đơn giản và nhanh hơn nhiều là dùng 1 câu SQL để cập nhật giá trị “token” cho Object User trong DB 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:

ClazzService

```
public Id create (Course courseld, Clazz clazz) throws Exception {
    Course course = get(courseld);
    Teacher teacher = clazz.getTeacher();
    if (course == null || teacher == null || !isTeacherAvailable(teacher)) {
        if (clazz.getStudents() > 40 && !course.isExpired()) {
            throw new BusinessException("bla bla");
        }
    }
}
```

www.tech3s-mentor.com

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 SW khi đưa đến User thì ngoài tính năng chạy ổn, đáp ứng tốt nhu cầu User thì hiệu năng hệ thống (**System Performance**) của SW cũng là vấn đề đáng lưu tâm. Để thấy giữa 2 SW tính năng tương đương nhau, một SW tính năng được đánh giá tốt hơn tí nhưng System Performance hơi kém thì sẽ gây không thiện cảm cho User.

r. Version Control System – VCS (Hệ điều khiển phiên bản)

Hệ quản trị phiên bản (**Version Control System – VCS**) là hệ thống quản lý Source Code, được phát triển bởi nhiều người theo các Version khác nhau giúp cho việc phát triển SW thuận tiện hơn nhiều.

s. Git Standard (Chuẩn Git)

- **Git – A distributed VCS** (Dạng chuẩn của 1 VCS phân tán): Là các đặc tả về 1 hệ VCS, các công ty sẽ hiện thực Git để đưa ra các bản khác nhau như GitHub, GitLab, BitBucket...
- **Git Repository** (Kho trên Git): Là nơi lưu trữ Source Code của 1 dự án.
- **Git Branch** (Nhánh trong Git): Là 1 bản chứa Source Code đầy đủ tách ra từ 1 Branch nào đó, khi nào làm xong sẽ nhập lại (**merge**) vào Branch đó. Khi khởi tạo 1 Repository sẽ có 1 Branch được tạo ra có tên là “**master**”.
- **Git Commands** (Các lệnh của Git):
 - **checkout** (Khởi tạo cục bộ): tạo 1 Local Branch (1 Branch ở máy cục bộ) từ 1 Remote Branch (1 Branch ở trên Repository).
 - **commit** (Lưu cục bộ): lưu những phần đã sửa ở trên 1 Local Branch vào 1 Commit trên Branch đó.
 - **push** (Đẩy lên): đẩy những Commit được tạo ra ở trên 1 Local Branch lên Remote Branch tương ứng trên Repository.
 - **fetch** (Kéo về): kéo những Remote Branch mới hoặc đã bị thay đổi từ Repository về các Local Branch tương ứng.
 - **merge** (Gộp vào): gộp 1 Local Branch A vào 1 Local Branch B.
 - **pull** (Kéo về và gộp lại): kéo những Remote Branch mới hoặc đã bị thay đổi từ Repository về máy cục bộ rồi tự động gộp lại với các Local Branch tương ứng
- **Pull Request** (Yêu cầu kéo về): là yêu cầu kéo nội dung của 1 Remote Branch vào 1 Remote Branch khác, cái này được tạo ra thường xuyên sau khi Software Developer – SD làm xong trên 1 Remote Branch nào đó muốn nhập vào Remote Branch có tên là “**develop**” được cả nhóm xây dựng.

t. Code Conflict (Xung đột mã)

Trong quá trình phát triển SW, việc các SD cùng sửa 1 hay nhiều File trên cùng 1 Repository xảy ra như chuyện hàng 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 SD tạo Pull Request sau phải có trách nhiệm xử lý xung đột. Trong quá trình xử lý có những đoạn Code nào do các SD 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:

- Khi User đang xài Version X, Scrum Team đang làm Version X+1, trong quá trình xài User báo một số lỗi gấp phải. Thé là các SD sẽ thay nhau xử lý các lỗi này trên Version X gọi là **HOTFIX**. Sau khi sửa lỗi xong và User đã xài ổn định Version X thì những phần sửa này sẽ được gộp lại vào Version X+1 để làm tiếp cho đến khi triển khai (**Release**).
- Lúc gộp này sẽ có thể gây ra xung đột giữa phần sửa trên Version X và phần SD đang phát triển trên Version X+1. Trường hợp này thì các SD có liên quan đến xung đột phải ngồi lại với nhau xử lý.

Việc Code Conflict xảy ra rất thường xuyên trong các Scrum Team, do vậy kỹ năng xử lý Code Conflict là cực kỳ quan trọng giúp cho phát triển SW nhanh hơn và hạn chế lỗi tốt hơn.

u. Self-Experience

Chọn 1 PL mà bạn thích như Java, C#, PHP, Ruby...

1. Viết 1 Program in ra câu “Hello World!”
2. Viết 1 Program trong phần Data Scope ở trên, chạy thử xem kết quả in ra màn hình là gì.
3. Viết 1 Program nhận vào 1 số nguyên dương, in ra màn hình các cấu trúc hình * sau:

```
Please enter n (n>0): 3
1. n*n star matrix:
* * *
* * *
* * *

2. Left star triangle:
*
*
*
*

3. Inversed Left star triangle:
*
*
*
*

4. Right star triangle:
*
*
*
*

5. Inversed Right star triangle:
*
*
*
*

6. Star pyramid:
*
*
*
*
*
```

4. Viết 1 Program về rút số trùng thưởng:
 - Cho trước 1 List chứa các con số nguyên từ 1-99, chiều dài List tùy ý.
 - Sinh ra ngẫu nhiên 1 con số từ 1-99 (rút số ngoài đời).
 - In ra 1 List các chỉ số của List ở trên mà có con số trùng với số sinh ngẫu nhiên.
 - Ví dụ: list = {3,6,66,9,66,23,35,36,66,80,90,99}, số ngẫu nhiên = 66 -> indexes = {3,5,9} (giả sử danh sách có chỉ số tính từ 1, một số PL xài chỉ số tính từ 0).

5. Viết 1 Program

- Cho trước 1 List chứa các con số thực (float) bất kỳ.
- Yêu cầu User nhập vào 1 con số thực (float).
- Báo cho User biết con số họ nhập vào có tồn tại trong List trên hay không. Sau đó hiện ra câu hỏi “Bạn có muốn tiếp tục không”? Nếu User nhập vào “N” -> thoát Program, nhập vào “Y” thì quay lại bước 2, nhập bất cứ gì khác “N” và “Y” thì hiện lại câu hỏi.
- Tạo 1 Class “Product” có các Property: “title”, “desc”, “price”, “quantity” & getters/setters.
- Tạo 1 List các Product Object, duyệt từng Product in ra màn hình.
- Tạo 1 Set các Product Object, duyệt từng Product in ra màn hình.
- Tạo 1 Class Node có Property: name & getter/setter. Xây dựng 1 cây nhị phân (1 Node có thể có Node trái/phải) có 3 Node: Dad -> Son & Daughter, duyệt cây in ra màn hình các Node.

6. Viết 1 Program

- Tạo 1 Abstract Class “Shape” có các Property: “color” & getter/setter, có các Abstract Method: “draw”, “calculateArea”.
- Tạo 1 Class “Circle” thừa kế “Shape” có các Property: “radius” & getter/setter, có các overriding Method: “draw”, “calculateArea”, và 1 overloading Method **“draw(String customColor);”**
- Tạo 1 Class “Square” thừa kế “Shape” có các Property: “edge” & getter/setter, có các overriding Method: “draw”, “calculateArea”.
- Tạo 1 Class “Rectangular” thừa kế “Shape” có các Property: “width”, “height” & getters/setters, có các overriding Method: “draw”, “calculateArea”.
- Tạo các Object từ các Class “Circle”, “Square”, “Rectangular”, viết 1 PROC **playWithShape(Shape shape);** thể hiện Interitance & Polymorphism.

7. Viết 1 Program

- Tạo 1 DB trong MySQL có tên là “programming”.
- Tạo 1 Table “products” trong DB “programming” có các Column: “id” (PK), “title” (VARCHAR), “desc” (VARCHAR), “price” (FLOAT), “quantity” (INT).

- Tạo 1 Class “ProductDao” với các Method:
 - *Product create(Product newProduct)*: tạo 1 Product dưới DB
 - *void read(String productId)*: đọc 1 Product từ DB
 - *Product update(Product updatedProduct)*: cập nhật Product xuống DB
 - *void delete(String productId)*: xóa 1 Product trong DB
 - *List<Product> search(String title)*: tìm kiếm các Product có “title” chứa từ khóa **title** nhập vào.
- Viết Code để gọi các Method của “ProductDao” và in kết quả ra màn hình.

8. Viết 1 Program

- Đọc File “products.csv” chứa thông tin của các Product từ HD lên bộ nhớ.
- Xử lý File rồi tạo List các Product
- Lưu List các Product xuống DB “programming” ở trên.
- Đọc các Product vừa lưu xuống DB lên rồi in ra màn hình.

9. Viết 1 Program

- Tạo 1 Class “MathLib”
- Tạo 1 Class Method *solveFirstEquation(float a, float b)*; để giải phương trình bậc 1.
- Tạo 1 Class Method *solveSecondEquation(float a, float b, float c)*; để giải phương trình bậc 2.
- Viết Code để sử dụng “MathLib” và in kết quả ra màn hình

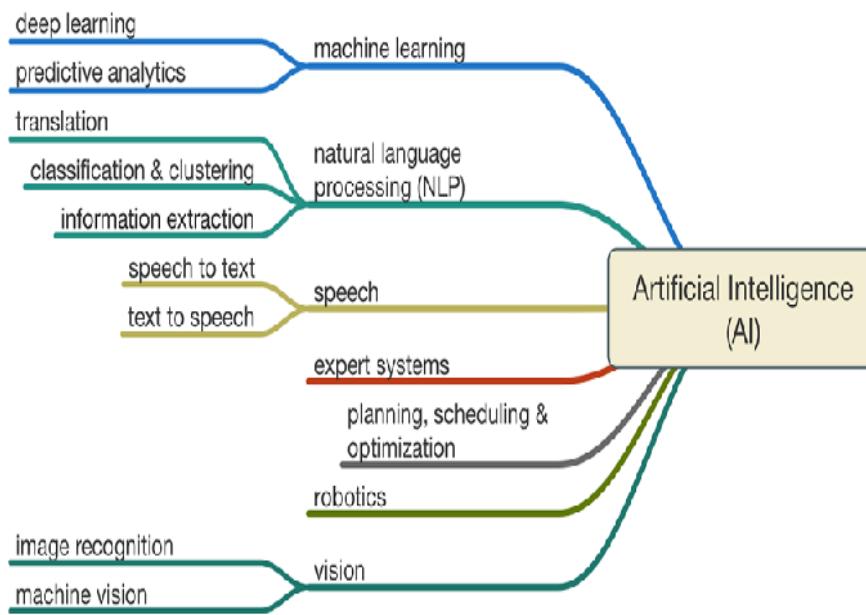
10. Viết 1 Program

- Tạo các Class “Customer”, “Order”, “OrderLine” (1 Order có 1 Customer và nhiều OrderLine).
- Tạo các Table “customers”, “orders”, “order_lines” trong DB “programming”.
- Tạo 1 Class “OrderDao” có các Method tương tự như “ProductDao”
- Tạo 1 List các Order
- Lưu List các Order này xuống DB
- Đọc List các Order từ DB lên rồi in ra màn hình

Ghi chú:

- Các câu sau có thể sử dụng lại Code của những câu trước.
- Có thể xem vài phần trả lời tham khảo: <https://github.com/homertruong66/tech3s-mentor>.

7) Artificial Intelligence – AI (Trí tuệ nhân tạo)



Trí tuệ nhân tạo (**Artificial Intelligence – AI**) là lĩnh vực trong đó con người giúp CPT có trí tuệ như con người để tận dụng sức mạnh của CPT xử lý những vấn đề mà con người không xử lý nổi do giới hạn về thời gian hoặc hạn chế về năng lực tính toán.

Tính chất lý tưởng của AI là khả năng suy luận và đưa ra giải pháp có cơ hội cao nhất đạt được một mục tiêu cụ thể.

a. AI Fundamental Formula (Công thức nền tảng của AI)

Công thức nền tảng của AI rất đơn giản:

AI = Presentation (Mô hình) & Search (Tìm kiếm)

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 CPT “học”, sau đó CPT sẽ tìm kiếm các giải pháp cho vấn đề của con người bằng các thao tác tìm kiếm các giải pháp trên mô hình đó.

Ví dụ: trong trò chơi Cờ tướng, bàn cờ được mô hình hóa bằng mảng 2 chiều cho CPT hiểu và CPT được cung cấp cách tìm nước đi bằng giải thuật Alpha-Beta cùng với một hàm lượng giá để đánh giá 1 thê cờ được bao nhiêu điểm. Sau mỗi nước đi của người thì CPT sẽ tìm 1 nước đi phù hợp nhất dựa trên trạng thái của bàn cờ hiện tại (hình vị trí các quân cờ đang nằm trên bàn cờ).

b. AI Applications (Các ứng dụng AI)

Số lượng ứng dụng AI là vô cùng nhiều, một số ứng dụng AI như:

- Choi Cờ (Cờ tướng, Cờ vua, Cờ vây...): dự án “XiangqiClub” tôi đã làm luận văn tốt nghiệp đại học ở Việt Nam.
- **Self-driving Car** (Xe tự lái)
- **Natural Language Processing** (Xử lý ngôn ngữ tự nhiên)
- **Speech Recognition** (Nhận dạng giọng nói)
- **Face Recognition** (Nhận dạng khuôn mặt)
- **Expert System** (Hệ chuyên gia): có thể dùng trong mảng y khoa, mảng xuất nhập khẩu (như dự án “GPM” tôi đã làm ở Canada).
- **Vision** (Phim ảnh): dùng trong ngành điện ảnh.
- **Unusual Debit Card Usage & Large Account Deposit** (Việc sử dụng thẻ ghi nợ bất thường & Nạp tiền số lượng lớn): dùng trong ngành tài chính/ngân hàng.
- **Recommendation System** (Hệ thống gợi ý): dùng trong ngành eCommerce.
- **Data Mining** (Khai thác dữ liệu): bài toán dự báo thời tiết, dự án “USPS” (User Satisfaction Prediction System) tôi đã làm luận văn tốt nghiệp Master of IT ở Canada.
- ...

c. Self-Experience

Viết một chương trình cờ tướng đơn giản.

8) Data Science – DSC (Khoa học dữ liệu)

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 – DSC**) sẽ giúp họ.

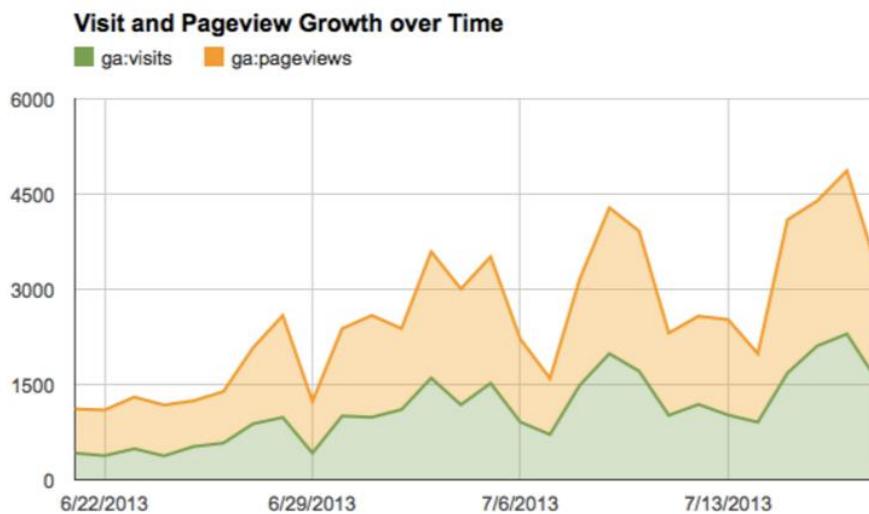
DSC gồm 3 nhánh được thể hiện trong hình sau:



a. Data Analytics (Phân tích dữ liệu)

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 eCommerce Web, có thể theo dõi những mặt hàng mà Customer hay tương tác mà chưa mua (như bỏ vô/ra giỏ hàng nhưng không lấy) để giảm giá/khuyến mãi một chút nhằm kích thích khách hàng mua các mặt hàng đó.

b. Data Mining (Khai thác dữ liệu)

(*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 $\mathbf{X} = (X_1, X_2, \dots, X_d)^T$, where X_1, X_2, \dots, X_d are d features of pattern \mathbf{X} and d is the number of dimensions of the feature vector \mathbf{X} .

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

Pattern Classification

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

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

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 °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).

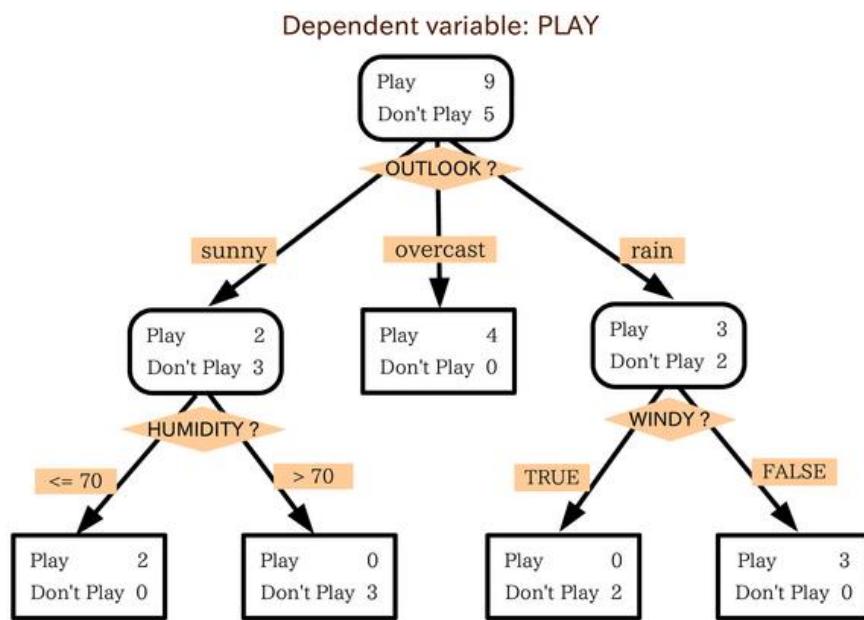
He wants to know if the customers will play or not in a given day, so the target value is a category class belonging to a set of classes: {Play, Don't play}.

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. Data Warehouse – DW (Kho dữ liệu)

Trong một doanh nghiệp lớn có nhiều phòng ban, cơ quan khác nhau. Mỗi nơi xài những SW quản lý với các DB khác nhau. Điều này dẫn đến việc chia sẻ thông tin giữa các nơi trở nên khó khăn hơn. Vì vậy kho dữ liệu (**Data Warehouse – DW**) được thiết kế để giải quyết vấn đề này.

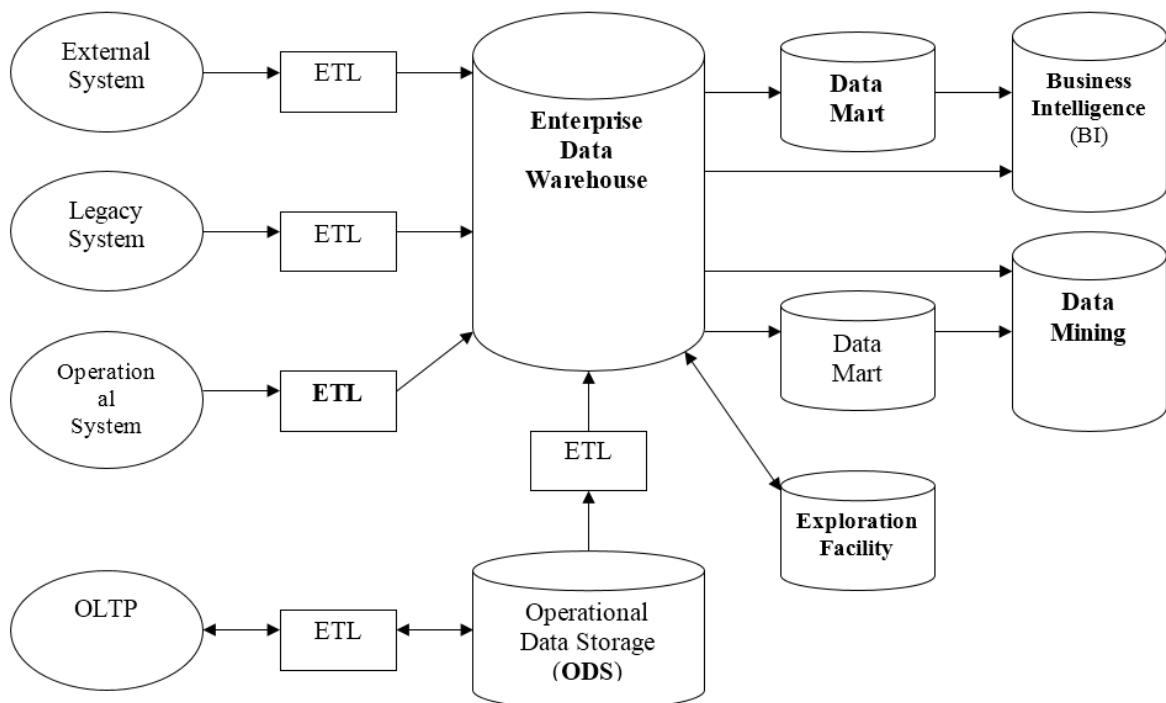
DW của doanh nghiệp chứa dữ liệu từ nhiều nguồn khác nhau như từ các phòng ban hoặc online, bao gồm cả dữ liệu cấu trúc (**Structured Data**) và Dữ liệu không cấu trúc (**Unstructured Data**). Dữ liệu từ nguồn nào đó vào DW qua quá trình Trích xuất-Chuyển đổi-Nạp (**Extract-Transfer-Load – ETL**).

DW có thể được xây dựng theo 2 cách:

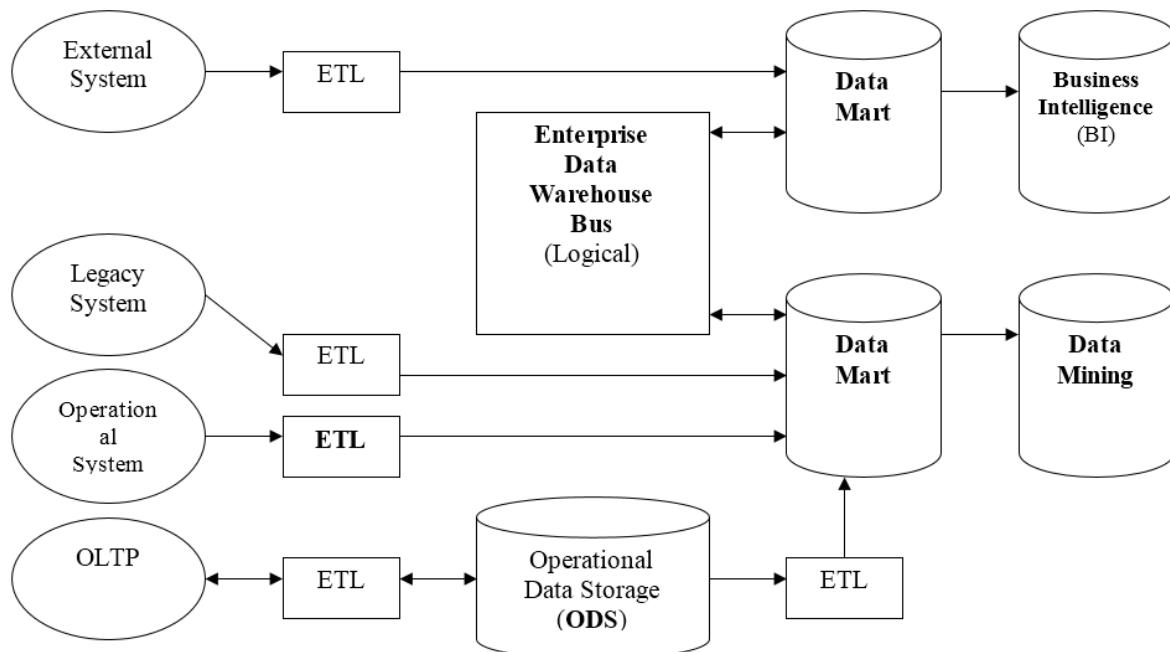
- **Top-down** (Từ trên xuống) theo William Inmon -> **Physical DW**
 - Theo cách thiết kế này thì DW được xây dựng từ lúc doanh nghiệp bắt đầu đi vào hoạt động. Dữ liệu hỗn tạp từ các hệ thống ngoài doanh nghiệp (**External System**), hệ thống đã tồn tại (**Legacy System**), hệ thống vận hành (**Operation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

Top-down DW & Bottom-up DW được hiển thị trong hình dưới đây.

Top-down DW



Bottom-up DW



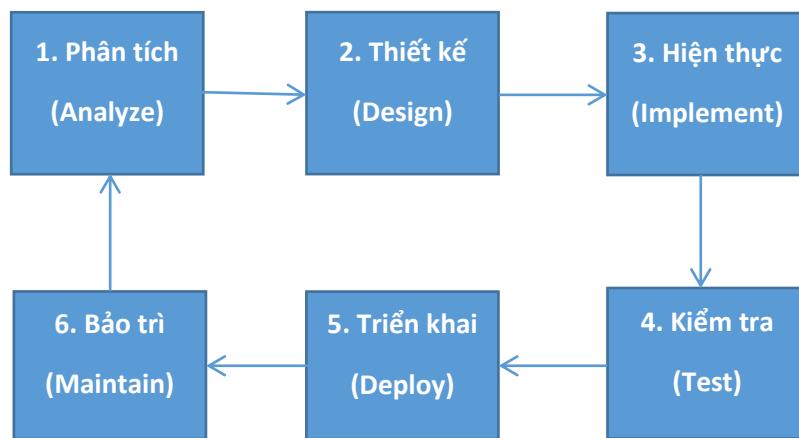
d. Self-Experience

1. Dụng 1 Website, sử dụng Google Analytics để thu thập dữ liệu rồi xem các báo cáo.
2. Tải công cụ WEKA và cuốn sách về học & hành: <https://www.cs.waikato.ac.nz/ml/weka/>

9) Software Engineering – SWEN (Công nghệ phần mềm)

Để phát triển SW tốt thì hiển nhiên chúng ta phải biết cách thức xây dựng SW hiệu quả hay còn gọi bằng từ chuyên môn hơn là công nghệ phần mềm (**Software Engineering – SWEN**).

Thế thì SWEN là gì vậy? Theo định nghĩa một cách hàn lâm thì SWEN hơi dài dòng khó hiểu. Nói nôm na dễ hiểu hơn thì SWEN đưa ra cách để chúng ta xây dựng 1 SW hiệu quả với 6 bước sau:



a. Software Development Life Cycle – SDLC (Vòng đời phát triển SW)

Các bước trên tạo thành vòng đời phát triển phần mềm (**Software Development Life Cycle – SDLC**) khép kín.

b. Software Development Process (Qui trình phát triển phần mềm)

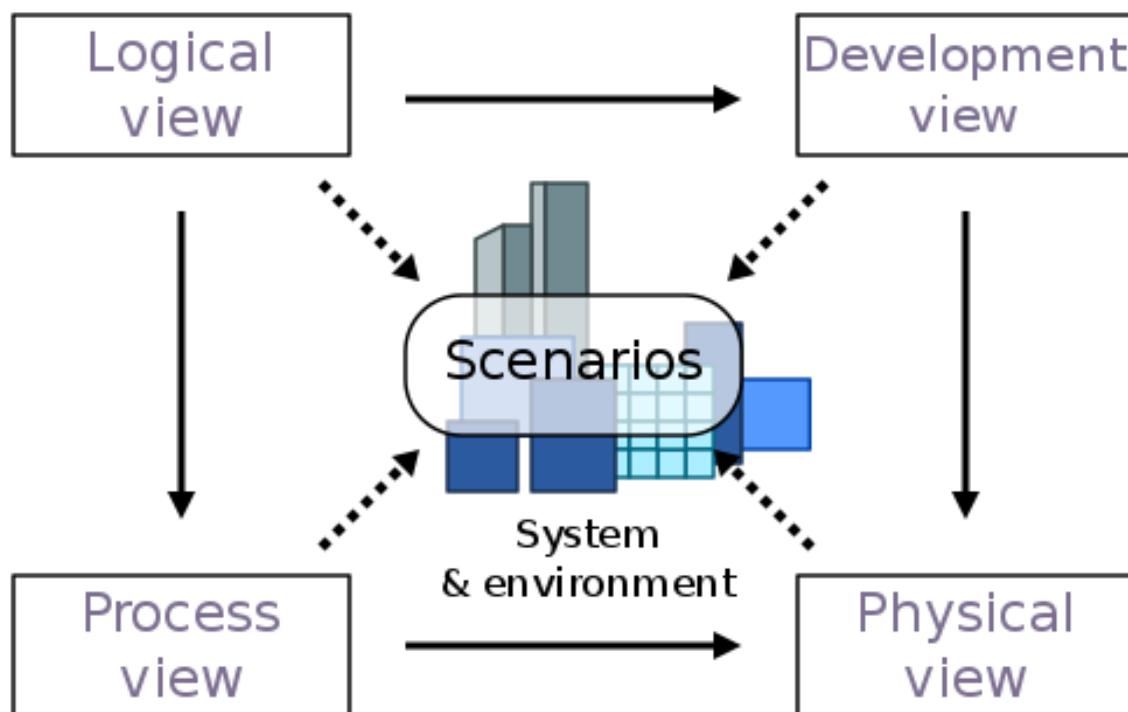
Từ thời SWEN mới ra đời, phương pháp phát triển phần mềm phổ biến là phương pháp thác nước (**Waterfall Methodology**), trong đó tất cả 6 bước trên được thực hiện tuần tự TRÊN TẤT CẢ TÍNH NĂNG SW. Các SW lúc trước có các yêu cầu chức năng (**Functional Requirement – FR**) biết trước là sẽ đáp ứng đúng nhu cầu của User, ví dụ như SW kế toán, SW quản lý nhân sự... nên Waterfall Methodology áp dụng xây dựng 1 SW như vậy mất khoảng 6 tháng đến 1 năm mới ra mắt User cũng được.

Tuy nhiên, sau này người ta thấy phát triển cả đồng tính năng SW 1 lèo suốt 6 tháng đến 1 năm rồi mới đưa User xài thì lỡ có tính năng gì đó User không thật sự muốn hoặc muốn chỉnh sửa gì đó thì chi phí thay đổi rất cao. Cho nên phương pháp xây dựng SW mới đã ra đời: đó là phương pháp nhanh & lặp (**Agile Methodology**). Với Agile Methodology thì các bước được thực hiện tuần tự TRÊN VÀI TÍNH NĂNG SW 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 Process**.

c. “4+1 Views” Model (Mô hình “4+1 Views”)

Xây dựng SW là công việc không thể của một người, mà là của nhiều người trong 1 Scrum Team, thế thì vấn đề đặt ra là làm sao để chia sẻ kiến thức về SW cho cả Scrum Team để toàn bộ thành viên đều cùng hiểu hệ thống như nhau, nhất là các thành viên mới tham gia vào Scrum Team?

Các chuyên gia phần mềm trên thế giới đã phát triển nhiều mô hình để chia sẻ kiến thức về hệ thống phần mềm cho các thành viên trong nhóm phát triển phần mềm, trong đó mô hình tổng quát nhất và được sử dụng thông dụng nhất là mô hình “4+1 Views”:



1. Góc nhìn luận lý (**Logical View**): thể hiện góc nhìn luận lý vào hệ thống thông qua mô hình lĩnh vực (**Domain Model**) và các lược đồ trạng thái (**State Diagram**) nếu có.
2. Góc nhìn quy trình (**Process View**): thể hiện góc nhìn hoạt động của hệ thống qua các luồng nghiệp vụ (**Business Flow**) biểu diễn bằng các lược đồ dòng chảy (**Flow Chart**).
3. Góc nhìn phát triển (**Development View**): thể hiện góc nhìn cách hệ thống được hiện thực thông qua mô hình thành phần (**Component Model**) và mô hình kiến trúc (**Architecture Model**).
4. Góc nhìn vật lý (**Physical View**): thể hiện góc nhìn cách hệ thống được triển khai vật lý qua mô hình triển khai (**Deployment Model**).
5. Góc nhìn tương tác người dùng – ngữ cảnh (**User Stories – Scenarios**): đặc tả những tương tác của người dùng với hệ thống trong các ngữ cảnh cụ thể.

d. Phân tích (Analyze)

Các FR của SW được **Business Analyst (BA)** phân tích để làm rõ các Business Flow, các qui luật nghiệp vụ (**Business Rule**), các tương tác giữa User với SW (**User Story – US**), giao diện người dùng (**User Interface – UI**), chuyển đổi UI (**UI Transition**) trong các US...

Phản 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.

e. Thiết kế (Design)

Dựa trên kết quả phân tích các FR, kiến trúc sư phần mềm (**Software Architect – SA**) cùng các thành viên SD, phát triển vận hành (**DevOps – DO**), và các thành viên khác trong ban duyệt kiến trúc (**Architecture Review Board – ARB**) sẽ thiết kế hệ thống IT (bao gồm SW).

Tùy theo mô hình phát triển SW được sử dụng mà các lược đồ thiết kế (**Design Diagram**) sẽ khác nhau.

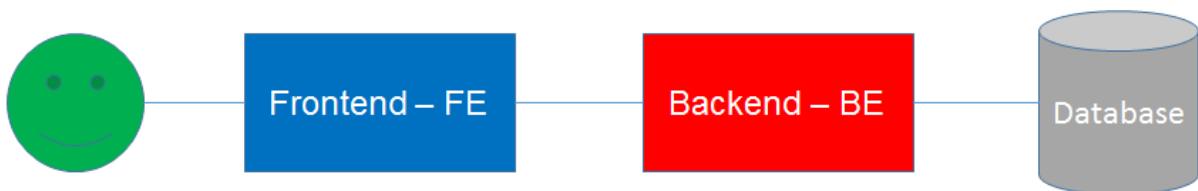
f. Hiện thực (Implement)

Vai trò SA dựa trên các FR và các yêu cầu phi chức năng (**Non-Functional Requirement – NFR**) để lựa chọn 1 chัng công nghệ (**Technology Stack**) phù hợp, rồi lập trình xây dựng 1 khung hệ thống (**System Skeleton**).

Tiếp đó, đội ngũ SD sử dụng System Skeleton và dựa trên phản phân tích các FR và các Design Diagram để lập trình xây dựng các tính năng phần mềm (**Software Function**).

Theo Scrum Process, mỗi Feature gồm nhiều US sẽ được hiện thực trong một/vài lần phát triển nhanh (**Sprint**). Một Sprint thường diễn ra từ 2-4 tuần tùy theo từng giai đoạn của dự án.

Mỗi US có thể bao gồm một hoặc nhiều UI và các UI Transition trong quá trình User tương tác với SW. Vì vậy hiện thực 1 US sẽ bao gồm 2 phần:



- Lập trình phía User (**Frontend Programming – FEP**): FE có thể là Web hoặc Mobile App, các UI được chuyển đổi theo 1 thứ tự được định nghĩa trước, dữ liệu thể hiện trên UI được lấy từ Backend.
- Lập trình xử lý nghiệp vụ SW (**Backend Programming – BEP**): xử lý toàn bộ nghiệp vụ SW, lưu dữ liệu vào DB, cung cấp dữ liệu cho FE.

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

g. Kiểm thử (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 Repository của dự án để kiểm tra từng Module của hệ thống xem có chạy đúng khi nó chạy một mình (không liên kết với các Module khác) hay không.

Việc hiện thực UT có giá trị rất lớn trong quá trình bảo trì SW 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 đó hay không.

h. Triển khai (Deploy)

Khi các US trong Sprint thứ i đã được hiện thực và kiểm tra ok hết rồi thì DevOps sẽ triển khai lên môi trường sử dụng (**Production Environment**) cho User xài.

i. Bảo trì (Maintain)

Sau khi User đã xài SW thì Scrum Team bắt đầu làm 2 việc song song: phát triển US mới & bảo trì hệ thống.

j. Self-Experience

Áp dụng SWEN với mô hình “4+1 Views” cho 1 ứng dụng eCommerce cơ bản.

C.Soft Skills (Kỹ năng mềm)

1) Professional English (Tiếng Anh làm việc)

Trong thế giới IT, English là cầu nối chúng ta đến với kho kiến thức IT không lò 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 English phục vụ cho công việc IT là điều bắt buộc đối với bất cứ dân IT nào.

Nhưng dân IT luyện English 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 từ lúc tôi mới tốt nghiệp đại học bước vào môi trường làm việc ngoài đời. Thực tế là tôi đã rất vất vả, nhiều lúc rất nản, trong việc luyện English để phục vụ công việc IT của tôi trong suốt nhiều năm.

Giờ đây tôi đã tổng hợp được một số kiến thức/kinh nghiệm luyện English phục vụ cho công việc IT với hi vọng sẽ giúp bạn nâng cao hơn kỹ năng English và tiết kiệm thời gian cho bạn:

- **Fundamental Knowledge** (Kiến thức nền tảng):

- **Nature of Word** (Bản chất của từ)
- **Function of Word** (Chức năng của từ)
- **Compound Noun** (Danh từ ghép)
- **Sentence** (Câu)
- **Tense** (Thì)

- **Essential Skill** (Kỹ năng thiết yếu):

- **Reading Skill** (Kỹ năng đọc)
- **Writing Skill** (Kỹ năng viết)

- **Complement Skill** (Kỹ năng bổ sung):

- **Listening Skill** (Kỹ năng nghe)
- **Speaking Skill** (Kỹ năng nói)

Chúng ta bắt đầu tìm hiểu một số kiến thức nền tảng English mà dân IT cần nắm.

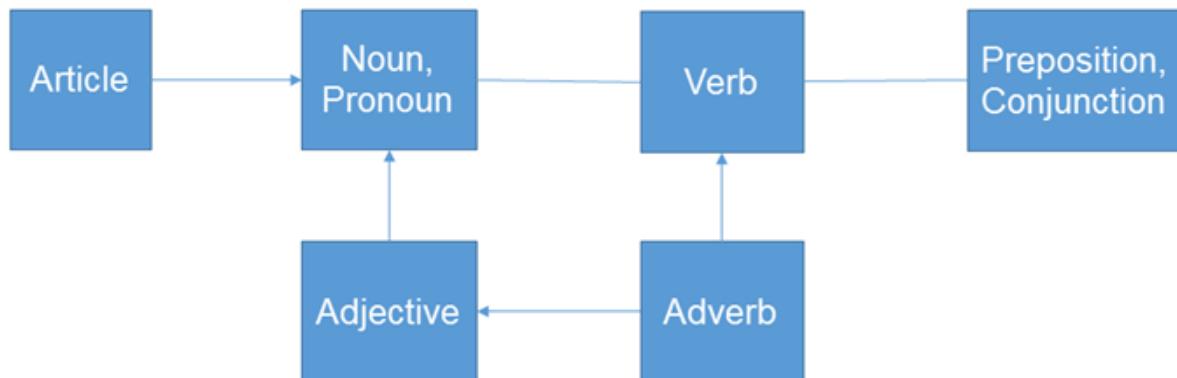
a. Nature of Word (Bản chất của từ)

Mỗi từ English sẽ có bản chất là một trong các loại từ sau:

- **Article** (Mao từ)
- **Noun/Pronoun** (Danh từ/Đại từ)
- **Verb** (Động từ)
- **Preposition/Conjunction** (Giới từ/Liên từ)
- **Adjective** (Tính từ)
- **Adverb** (Trạng từ)

Các loại từ này có quan hệ với nhau, bù nghĩa cho nhau, được đặc tả trong hình dưới đây, mũi tên A -> B thể hiện A bù nghĩa cho B, ví dụ: Article bù nghĩa cho Noun/Pronoun, Adjective bù nghĩa cho Noun/Pronoun, Adverb bù nghĩa cho Adjective & Verb.

Nature of Word



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 bù nghĩa cho nhau của các từ trong hình trên sẽ giúp chúng ta đọc hiểu đúng cũng như viết đúng, ví dụ:

- “**Make things different.**” -> ở đây chữ “different” là Adjective bù nghĩa cho chữ “things” là Noun nên câu này nghĩa là “Hãy làm những điều khác biệt” (tức là làm những điều gì đó mới mẻ).

- “**Make things differently.**” -> ở đây chữ “differently” là Adverb bổ nghĩa cho chữ “Make” là Verb nên câu này nghĩa là “Hãy làm những điều đó bằng các cách khác nhau”.
-> Hiển nhiên nghĩa của 2 câu này khác nhau do sử dụng loại từ khác nhau.

Một lưu ý với loại từ là Adjective có đuôi là “ing” hoặc “ed”:

- **-ing** mang nghĩa chủ động, ví dụ: She is **boring** (Cô ấy thì chán lấm, ý là cô ấy không có gì thú vị để tiếp xúc hay hẹn hò).
- **-ed** mang nghĩa thụ động, ví dụ: She is **bored** (Cô ấy trông buồn chán, chắc cô ấy đang có gì đó không vui).

b. Function of Word (Chức năng của từ)

Mỗi từ English có 1 bản chất duy nhất, nhưng khi đứng trong 1 câu thì nó có thể có nhiều chức năng. Việc hiểu được chức năng của từ sẽ giúp chúng ta sử dụng đúng từ khi viết.

Ví dụ: Noun có thể có chức năng Subject (Chủ ngữ) hoặc Object (Tân ngữ) như hình sau.

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

c. Compound Noun (Danh từ ghép)

Các từ ghép trong tiếng Việt và English đều được viết từ trái qua phải (một số ngôn ngữ khác viết từ phải qua trái). Tuy nhiên, từ ghép tiếng Việt được hiểu từ trái qua phải còn từ ghép English hiểu từ phải qua trái. Điều này gây ra rất nhiều khó khăn cho người Việt Nam.

Ví dụ:

- “nhiệm vụ quan trọng cực kỳ” -> hiểu từ trái qua phải
- “the extremely important task” -> hiểu từ phải qua trái có nghĩa là “nhiệm vụ quan trọng cực kỳ”
- “management information system” vs “system information management” -> 2 từ ghép English này hiểu từ phải qua trái sẽ là “hệ thống thông tin quản lý” vs “quản lý thông tin hệ thống”, từ đầu nói đến “hệ thống” còn từ sau nói về “việc quản lý”.

Trong các loại từ ghép trong English, danh từ ghép (**Compound Noun**) rất là quan trọng. Các Compound Noun trong English rất đa dạng, hiểu được các cách ghép là chìa khóa giúp chúng ta hiểu được các câu dài và phức tạp.

Tựu trung có 1 số các công thức của Compound Noun 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.

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).

Các công thức trên thật ra là mở rộng của 2 nhóm danh từ ghép cơ bản:

- Adjective + Noun: công thức 1,2,3
- Noun + Noun: công thức 4,5

d. Sentence (Câu)

English có 3 loại câu:

1. **Simple Sentence** (Câu đơn): Chỉ có 1 **Clause** (Mệnh đề)
2. **Compound Sentence** (Câu ghép): Do nhiều Clause độc lập ghép lại.
3. **Complex Sentence** (Câu phức): Do các Clause chính & phụ tạo thành.

Simple Sentence có 9 loại được liệt kê trong hình sau:

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**.

Ghi chú: DO = Direct Object, IO = Indirect Object, Gerund = V-ing

Trong **Compound Sentence**, các Clause độc lập được kết nối với nhau bởi 3 Conjunction thông dụng là: “**and**”, “**but**”, “**so**”.

Compound Sentence

1. Developers implement features, **and** QC's 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.

Trong **Complex Sentence**, Clause phụ có 3 loại:

1. **Adjective Clause** (Mệnh đề tính từ)
2. **Noun Clause** (Mệnh đề danh từ)
3. **Adverb Clause** (Mệnh đề trạng từ)

Phần tóm tắt các điểm quan trọng của 3 loại Clause này được thể hiện trong các hình ở bên dưới đây.

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.

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.

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** 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.

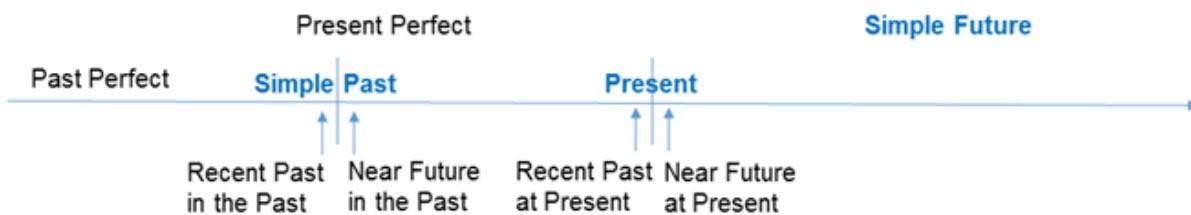
e. Tense (Thì)

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”. Đôi khi chúng ta cũng không cần thêm từ “đã” hoặc “sẽ” mà dùng Adverb để diễn đạt thì, ví dụ: “**Hôm bữa** tôi chả biết sao mà chương trình của tôi đang chạy tự nhiên đứng” -> đây là 1 câu trong thì quá khứ nhưng không có từ “đã”, thay vào đó Adverb “Hôm bữa” diễn đạt nghĩa trong quá khứ.

Nói đến thì (Tense) trong English thì chắc ai từng học qua cũng phải mệt não vì có quá nhiều loại Tense và cách dùng chúng cũng quá phức tạp, đôi khi cách dùng lại trùng lắp lên nhau nên có tình huống chả biết dùng Tense nào là hợp lý nhất.

Sau nhiều trải nghiệm thương đau, tôi đã tổng hợp các Tense được xài phổ biến một cách đơn giản hơn qua lược đồ sau:

Tense



1. Action in Future: Simple Future: Will/Shall + V1 (Infinitive)

2. Action in Present:

- a) Present : V1
- b) Near Future at Present : Am/Is/Are going to + V1
- c) Recent Past at Present : Have/Has just + V3 (Past Participle)
- d) Present Perfect : Have + V3

3. Action in Past

- a) Simple Past : V2 (Past)
- b) Near Future in the Past : Was/Were going to + V1
- c) Recent Past in the Past : Had just +V3
- d) Past Perfect : Had + V3

- Nếu hành động xảy ra trong tương lai, chúng ta cứ xài thì tương lai đơn cho đơn giản, các thì tương lai khác rất ít khi được xài.
- Nếu hành động xảy ra ở hiện tại, chúng ta lấy hiện tại làm mốc thời gian cho hành động:
 - Nhích tới 1 tí chúng ta có thì tương lai gần ở hiện tại (**Near Future at Present**)
 - Lùi về 1 tí chúng ta có thì quá khứ gần ở hiện tại (**Recent Past at Present**)

- 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 hành động không rõ thời gian. Ví dụ: “**I have worked on the project for 3 years.**” -> “Tôi đã làm dự án đó được 3 năm.” (giờ tôi vẫn còn làm trên dự án đó); “**She has been the good girl this year, so she deserves a Chrismast gift.**” -> “Cô ấy đã là cô gái tốt suốt năm nay nên cô ấy xứng đáng một món quà Giáng sinh.” (giờ cô bé vẫn tốt nhưng không biết cô bé bắt đầu tốt khi nào, có thể trước đó không tốt).
- Nếu 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 (**Near Future in the Past**), quá khứ gần trong quá khứ (**Recent Past in the Past**) và cuối cùng là quá khứ hoàn thành (**Past Perfect**).

f. Reading Skill (Kỹ năng Đọc)

Để đọc English nhanh và hiểu tốt, chúng ta cần thực hành thuần thục các bước sau:

1. Mỗi lần đọc 1 câu
2. Nhận dạng các cụm từ ghép -> hiểu nghĩa của nó từ phải qua -> xác định chức năng
3. Phân tích cấu trúc câu

Ví dụ: Java is a **general-purpose programming language** that is **class-based, object-oriented**, and designed to have as few **implementation dependencies** as possible.

-> Các cụm từ ghép là:

- “**general-purpose programming language**” -> Ngôn ngữ lập trình mục đích tổng quát
-> Subject Complement.
 - “**class-based**” -> dựa trên lớp -> bồ nghĩa cho cụm từ trên
 - “**object-oriented**” -> hướng đối tượng -> bồ nghĩa cho cụm từ trên
 - “**implementation dependencies**” -> các phụ thuộc hiện thực -> Direct Object
- > Cấu trúc câu: Complex Sentence – Adjective Clause

Thời gian đầu chúng ta sẽ làm các bước này với tốc độ rất chậm, sau khi quen thì tốc độ sẽ nhanh lên từ từ rồi sẽ đạt được sự nhuần nhuyễn.

Khi đọc English chúng ta không nên gặp bất kỳ từ mới nào cũng đi tra từ điển mà nên tập thói quen đoán nghĩa các từ mà chúng ta chưa biết, nếu đoán không ra mà những từ này đóng vai trò quan trọng trong việc hiểu nghĩa cả câu thì mới phải tra từ điển. Những từ nào chúng ta gặp nhiều lần trong 1 bài hoặc trong 1 ngày thì chúng ta cũng nên tra từ điển.

Tốt nhất là dùng từ điển Anh-Anh-Việt. Một trong những cuốn tự điển Anh-Anh-Việt online miễn phí khá hay là tự điển này: <https://www.thefreedictionary.com>.

Mỗi lần tra từ điển nên tra luôn cả gia đình từ (**Word Family**) thay vì tra mỗi một từ, ví dụ khi tra từ “collect” -> tra luôn các từ anh chị em của nó là “collective”, “collection”…

Những cụm từ nào mà chúng ta thấy phổ biến/quan trọng thì nên lưu vào 1 File, kèm theo ngữ cảnh (cái câu mà cụm từ này xuất hiện khi chúng ta đọc) chứ đừng để cụm từ riêng lẻ.

g. Writing Skill (Kỹ năng Viết)

Là dân IT thì chúng ta không bắt buộc phải viết English cho hay, mà phải viết cho đúng chính tả & ngữ pháp để người đọc hiểu đúng ý chúng ta, nếu viết hay được nữa thì càng tốt.

Để viết English cho đúng thì chúng ta phải tuân thủ những phần đã tìm hiểu ở trên:

- Viết đúng các cấu trúc câu
- Dùng đúng các thì
- Dùng đúng bản chất các từ
- Viết đúng những cụm từ ghép -> phần này cần phải đọc English nhiều để biết nhiều cụm từ ghép xài trong IT để diễn đạt được ý của mình, đôi khi có những ý được diễn tả rất dài dòng do không biết xài cụm từ ghép, ví dụ: ý “các tác vụ tốn nhiều thời gian”
 - Không xài cụm từ ghép: “the tasks that take a lot of time” -> dài dòng
 - Xài cụm từ ghép: “time-consuming tasks”
- Viết đúng chính tả mỗi từ, nên dùng công cụ kiểm tra lỗi chính tả English

h. Listening Skill (Kỹ năng Nghe)

Kỹ năng nghe English quan trọng khi bạn làm việc trực tiếp với người nước ngoài hoặc tham dự các hội thảo bằng English.

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 English 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 (**Idiom/ Expression**) và các từ lóng (**Slang Word**).
- 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 CÓ NÓI với tốc độ bên ngoài thực tế khá là nhanh và nuốt chữ nhiều.
- Chúng ta thiếu từ vựng thực tế (**Real-life Vocabulary**) nên không biết những từ họ nói phát âm như thế nào thì làm sao nhận biết được chúng khi nghe.

Do đó, để luyện nghe English cho công việc thì chúng ta phải học English thực tế!!!

Tôi đã từng thử nghiệm 1 chương trình học English thực tế 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 chúng ta phải **rất kiên trì và khổ luyện trong thời gian khá dài...**

i. Speaking Skill (Kỹ năng Nói)

Kỹ năng English cuối cùng và cũng khó nhất đó là nói English (**English Speaking**).

Thú thật là từ lúc học English 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 English.

Tôi nghĩ không chỉ riêng tôi mà hầu hết người Việt Nam nói riêng và những người học English trên thế giới như Hàn Quốc, Trung Quốc, các nước Nam Mỹ... nói chung đều ngại nói English. Tại sao vậy?

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

- Sợ nói sai thì người bản xứ (**Native Speaker**) 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 English hàng ngày nên dễ bị cứng miệng.
- Không có đủ Vocabulary để diễn đạt ý mình muốn.

Tôi có một số lời khuyên giúp bạn cải thiện kỹ năng nói English:

- **Hãy nói English với suy nghĩ English trong đầu,** đừng suy nghĩ tiếng Việt sau đó dịch ra English trong đầu rồi mới nói thì không kịp thời gian đâu; thứ hai là suy nghĩ bằng English giúp “cảm” được ngôn ngữ/văn hóa, gọi là sắc thái của ngôn ngữ (**Language Nuance**). Trải nghiệm thực tế của tôi khi học tiếng Pháp ở Canada thì lúc đầu tôi suy nghĩ English 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 English/Francais hay hoặc bay bổng - miễn đối phương hiểu đúng ý), **và không sợ sai** bằng cách thực hành nói English hàng ngày trước gương, đừng quan tâm quá nhiều đến ngữ pháp (văn nói với văn viết khác xa nhau lắm), và câu cú này nọ sẽ làm chậm phản xạ của chúng ta khi nói.
- **Hãy tìm mọi cách nói English càng nhiều càng tốt với Native Speaker**, hạn chế nói English với người không phải bản xứ (**Non-Native Speaker**), và **bắt chước** thật nhiều phát âm, từ vựng cũng như cách nói của Native Speaker để sau này nói với Native Speaker khác thì khả năng họ hiểu sẽ rất cao.
- **Tích lũy Real-life Vocabulary 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.

j. Popular Misunderstandings (Các hiểu lầm phổ biến)

Văn phong khác nhau giữa English & tiếng Việt gây ra 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? → Ủ (Tôi không làm) vs Không (Tôi làm).

→ **Cách trả lời của English 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 cái việc đã tắt server (server đã bị tắt nhưng người tắt quên bêng việc tắt này → server không còn chạy).

k. Self-Experience

1. Mỗi ngày dành 1 tiếng ra đọc sách hay bài viết IT bằng tiếng Anh, lưu những cụm từ ghép vào 1 File cùng ngữ cảnh nó xuất hiện, cuối câu ghi nghĩa bằng tiếng Việt. Ví dụ:
 - A **programming language** is a formal language, which comprises a set of instructions that produce various kinds of output. -> “**Ngôn ngữ lập trình**”
 - Java is a **general-purpose programming language** that is class-based, object-oriented, and designed to have as few implementation dependencies as possible -> “**Ngôn ngữ lập trình mục đích tổng quát**”.Mỗi tuần xem lại các cụm từ ghép này từ đầu File đến cuối cùng. Lặp lại quá trình này suốt vài tháng đến 1 năm rồi xem hiệu quả thế nào.
2. Cố gắng viết càng nhiều câu càng tốt có sử dụng những cụm từ ở trên để diễn đạt ý gì đó.
3. Kiên trì nghe Real English hằng ngày.

2) **Logical Thinking** (Suy nghĩ luận lý)

a. Introduction

Mọi sự việc xảy ra trong cuộc sống hàng ngày đều tuân theo những qui luật nào đó. Cho nên khi chúng ta làm việc gì cũng cần suy nghĩ đến tính hợp lý (**Logic**).

Nói một cách khác là chúng ta suy nghĩ luận lý (**Logical Thinking**) một vấn đề nào đó dựa trên các dữ kiện (**Fact**), kiến thức (**Knowledge**), và kinh nghiệm (**Experience**) của chúng ta để đưa ra được một kết luận (**Conclusion**) hay một giải pháp (**Solution**) nào đó nhằm giải quyết các vấn đề (**Problem**) chúng ta đang gặp phải.

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

Ứng dụng của Logical Thinking 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 Logical Thinking thường dùng nhất là: Phát biểu điều kiện (**If Statement**).

Ví dụ: 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 này tự nhiên ngưng hoạt động, chúng ta có thể suy luận ra là có ai đã xóa cái file “xyz.sys”. Tuy nhiên, có thể có nhiều nguyên nhân khác làm cho hệ thống IT này ngưng hoạt động, nhưng ít nhất chúng ta tìm được 1 nguyên nhân có thể xảy ra nhờ dữ kiện chúng ta đã biết.

Một ví dụ khác thể hiện tính bắt đầu 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.

Logical Thinking là 1 quá trình 5 bước:

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

Logical Thinking giúp:

- BA nắm bắt yêu cầu SW & nghiệp vụ tốt sẽ phân tích FR được rõ ràng.
- SD hiểu được FR rõ sẽ hiện thực ít bị lỗi liên quan đến nghiệp vụ.
- QC hiểu được FR rõ sẽ kiểm thử hiệu quả hơn, đảm bảo các nghiệp vụ được đáp ứng đầy đủ nhất.

b. Self-Experience

1. Hãy suy nghĩ nghiệp vụ: 1 người muốn đăng ký tài khoản để sử dụng 1 SW mà SW muốn đảm bảo là họ nhập email phải chính xác là của họ?
2. Hãy suy nghĩ nghiệp vụ: 1 khách hàng mua hàng online ở một trang web thương mại điện tử sẽ diễn ra như thế nào?
3. Hãy suy nghĩ nghiệp vụ: vận chuyển 1 đơn hàng của 1 khách hàng trên 1 trang web thương mại điện tử sử dụng dịch vụ của 1 nhà vận chuyển (Fedex or DHL) ra sao?

3) Problem Solving (*Giải quyết vấn đề*)

a. Introduction

Hàng ngày từ lúc thức dậy cho đến lúc 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 (**Problem Solving**) như thế nào tốt nhất cũng là 1 bài toán không đơn giản, đòi hỏi chúng ta phải luôn rèn luyện kỹ năng giải quyết vấn đề.

Phần lớn các bạn trẻ IT Việt Nam không nhận thức được kỹ năng giải quyết vấn đề, nên khi gặp 1 vấn đề là họ nhảy vô xử lý liền với giải pháp đầu tiên mà họ nghĩ ra. Vì vậy rất nhiều trường hợp là chưa/không giải quyết được vấn đề hiện tại mà lại có thể gây ra những vấn đề mới vì giải pháp họ đã làm không phải là phù hợp nhất.

Problem Solving là 1 quá trình 5 bước:

1. Phân tích tất cả dữ kiện liên quan đến vấn đề để tìm ra nguyên nhân chính (**Root Cause**) của vấn đề.
2. Tìm tất cả giải pháp (**Solution**) có thể xử lý Root Cause này.
3. Với mỗi giải pháp tìm được, liệt kê tất cả ưu/nhược điểm (**Pros/Cons**) của nó.
4. So sánh & cân nhắc các giải pháp dựa trên Pros/Cons của chúng.
5. 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).

Mỗi lần giải quyết xong 1 vấn đề, một thời gian sau chúng ta cần ngâm lại xem giải pháp mà chúng ta đã chọn có phải là PHÙ HỢP NHẤT chưa? Có giải pháp nào tốt hơn nữa không? bởi vì có thể lúc ta chọn giải pháp đó trong điều kiện thiếu thông tin hoặc hạn chế về thời gian.

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

b. Self-Experience

Khi gặp 1 vấn đề, hãy áp dụng qui trình 5 bước trên để giải quyết nó.

4) Time Management (*Quản lý thời gian*)

a. Introduction

Mỗi ngày một dân IT có 8 tiếng làm việc trong Scrum Team, trong đó 85-88% thời gian trên được xem là có năng suất (**Productivity**), tức là có đóng góp cho sự phát triển của SW, phần còn lại cho các hoạt động khác như họp hành, thảo luận nhóm, hoạt động cá nhân khác...

Trong thời gian năng suất, dân IT 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 của Task 1.
- Sau đó SD A phải sửa xong 1 con Bug để DevOps C triển khai lên môi trường thử nghiệm (**Stage Environment**) để QC D kiểm thử.
- Rồi chiều đến thì SD A phải hiện thực 1 US cho DevOps C triển khai lên cho BA E chạy thử xem thế nào...

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

Từ kinh nghiệm thực tế, tôi chia sẻ một vài gợi ý giúp quản lý thời gian 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.
- Với SD, việc dễ làm trước cho Code vào Repository để hạn chế Code Conflict với các thành viên khác trong Scrum Team.
- Khi gặp một vấn đề hóc búa thì chỉ nên bỏ từ 1-3 tiếng tìm Root Cause 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 hoặc làm việc khác dễ hơn rồi quay lại lõi 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.

b. Elsenhower's Matrix (Ma trận của Elsenhower)

Đối với các vai trò liên quan đến quản lý thì có thể tham khảo ma trận Elsenhower:

	Urgent	Non-Urgent
Important	DO	PLAN
Unimportant	DELEGATE	ELIMINATE

- Urgent & Important:** Những việc vừa gấp vừa quan trọng thì phải bỏ thời gian làm ngay.
- Urgent & Unimportant:** Những việc gấp nhưng không quan trọng thì ủy quyền cho các thành viên khác làm.
- Non-Urgent & Important:** Những việc không gấp mà quan trọng thì lên kế hoạch để làm.
- Non-Urgent & Unimportant:** Những việc vừa không gấp vừa không quan trọng thì cần được loại bỏ.

c. Self-Experience

Hãy bắt đầu quan tâm đến việc quản lý thời gian trong việc học & làm hàng ngày.

5) Team Work (Làm việc nhóm)

a. Introduction

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 SW cũng rất đúng. Xây dựng 1 SW mà chỉ có 1 người làm thì SW đó sẽ được xây dựng nhanh nhưng không thể lớn và giúp con người giải quyết những việc to tát được.

Những SW 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 SW đó cho mạnh lên để đem đến các tính năng đáp ứng nhu cầu của nhiều User.

Như vậy, khi đã làm SW thì hiển nhiên dân IT sẽ làm trong một nhóm. Thì kĩ 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 kĩ 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 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 Process 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 (**Entrepreneur 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.

b. Self-Experience

Thực hành kỹ năng học nhóm/làm việc nhóm tuân thủ các điều trên càng nhiều càng tốt.

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

a. Introduction

Trong quá trình làm việc, rất nhiều lúc dân IT sẽ phải thuyết trình (**Presentation**) 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 đảm nhận vai trò mentor thuyết trình huấn luyện cho các bạn mới vào nhóm.

Khó hơn một mức nữa là dân IT phải thuyết trình phục khách hàng bên ngoài sử dụng giải pháp/sản phẩm của công ty mình.

Do đó, dân IT 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 minh họa 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 (**Communication**) với các thành viên làm các nghề IT khác nhau trong nhóm như BA, QC, DevOps... 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 IT trong cuốn cẩm nang “Việc làm trong ngành IT” của tôi).

b. Self-Experience

Thực hành thuyết trình & giao tiếp trong học nhóm/làm việc.

D.Advanced Knowledge (Kiến thức nâng cao)

Trong phần kiến thức nâng cao này, tôi sẽ chia sẻ một số phần sau:

1. **Software Architecture – SWAR** (Kiến trúc phần mềm): phần này cung cấp những khái niệm liên quan đến việc xây dựng/nắm bắt các loại kiến trúc phần mềm của hệ thống IT cũng như phần Backend & Frontend.
2. **Backend Programming – BEP** (Lập trình phía Backend): phần này cung cấp những khái niệm mà các BE Dev phải nắm vững và những mục nêu/ cần/ phải làm khi lập trình phía Backend.
3. **Frontend Programming – FEP** (Lập trình phía Frontend): phần này cung cấp những khái niệm mà các FE Dev phải nắm vững và những mục nêu/ cần/ phải làm khi lập trình phía Frontend.
4. **Non-Functional Requirement – NFR** (Yêu cầu phi chức năng): ngoài những FR mà hệ thống IT cung cấp cho User, phần này giới thiệu những yêu cầu phi chức năng cũng đóng vai trò rất quan trọng giúp cho hệ thống IT chạy ổn định và đáp ứng tốt các trải nghiệm của User cũng như vấn đề bảo mật.
5. **Reverse Engineering – REN** (Kỹ thuật học ngược): phần này cung cấp kỹ thuật để tìm hiểu 1 hệ thống IT đã được phát triển trước đó (**Legacy System**) mà hiện tại có thể đã ngưng phần phát triển hoặc vẫn đang được bảo trì/tiếp tục phát triển tính năng mới. Đặc điểm của hệ thống IT dạng này là tài liệu về phân tích/thiết kế/hiện thực của dự án không được nhóm phát triển trước đó tạo ra đủ (có thể do hạn chế về thời gian) để cho những người mới có thể hiểu được hệ thống làm cái gì (WHAT), như thế nào (HOW), tại sao lại như vậy (WHY).
6. **CSI Formula** (Công thức CSI): phần này giới thiệu một công thức giúp cho dân IT có thể nắm bắt công nghệ nhanh hơn.

1) Software Architecture – SWAR (Kiến trúc phần mềm)

Kiến trúc phần mềm (**Software Architecture – SWAR**) là nền móng để SW được xây dựng trên đó. Cho nên SWAR càng vững thì SW sẽ chạy ổn định và mở rộng tốt, ngược lại SWAR không tốt thì SW sẽ chạy rất thiếu ổn định và cực kỳ khó khăn khi thêm càng nhiều tính năng cũng như bảo trì.

Không có một loại SWAR nào là tốt nhất cho mọi SW. Tùy theo yêu cầu của SW mà sẽ có một loại SWAR phù hợp nhất. Tuy nhiên, sự phù hợp nhất ở đây cũng mang tính tương đối, có thể một SWAR được lựa chọn là phù hợp nhất lúc bắt đầu xây dựng SW, nhưng khi SW càng phát triển lên với nhiều sự thay đổi so với ban đầu để đáp ứng nhu cầu User thì SWAR này không còn phù hợp nữa, nhiều lúc phải bỏ đi để chọn SWAR khác làm lại từ đầu.

Trong phần tiếp theo chúng ta sẽ tìm hiểu khái quát một số SWAR và các khái niệm quan trọng liên quan đến SWAR từ trước đến hiện nay.

a. Client-Server Model (Mô hình Khách-Chủ)

Đây là một SWAR nền tảng cho các loại SWAR khác, trong đó một chương trình đóng vai trò là Server cung cấp các dịch vụ để một chương trình đóng vai trò là Client kết nối tới sử dụng dịch vụ. Một ví dụ điển hình của Client-Server Model là ứng dụng trò chuyện (**Chat Application**) trong đó Server là cầu nối giúp các Client gửi thông điệp cho nhau.

b. Desktop Application Architecture (Kiến trúc ứng dụng Desktop)

Đây là kiểu SWAR dựa trên Client-Server Model của các SW giai đoạn trước khi công nghệ Web ra đời. Loại SWAR này thường có các phần sau:

- Desktop Client: UI là các cửa sổ chạy trên nền Desktop của các OS như MS Windows, Mac OS, Linux... cho User tương tác.
- Server: đơn giản nhất và cũng phần lớn là Database Server hoặc 1 chương trình Server xử lý nghiệp vụ gì đó, có thể kết nối đến Database Server.

c. Model-View-Controller – MVC Pattern (Mẫu MVC)

MVC Pattern được phát minh để sử dụng trong các Desktop Application mục đích là tách riêng 3 thành phần chính của 1 ứng dụng nhằm mục đích tái sử dụng Code (**Code Reuse**) & phục vụ phát triển song song (**Parallel Development**).

1. **Model:** nơi chứa dữ liệu mà User thao tác, độc lập với các UI mà User tương tác, và các logic khác.
2. **View:** là UI hiển thị dữ liệu chứa trong Model dưới các hình thức khác nhau cho User xem, một Model có thể được hiển thị bởi nhiều View, và một View cũng có thể chứa dữ liệu của nhiều Model khác nhau.
3. **Controller:** nơi điều khiển Model & View.
 - Khi User chỉnh sửa dữ liệu trên View thì Controller sẽ cập nhật Model. Ví dụ: User thay đổi thông tin của 1 Product trên View và bấm nút “Save” thì Controller sẽ cập nhật dữ liệu về Product đó trong Model.
 - Khi Model bị thay đổi thì Controller cũng sẽ cập nhật các View tương ứng đang sử dụng dữ liệu của Model đó. Ví dụ: User thay đổi thông tin của 1 Product trên View 1 (ProductEditUI) và bấm nút “Save” thì Controller sẽ cập nhật dữ liệu về Product đó trong Model rồi cập nhật dữ liệu trên View 2 (ProductBrowsingUI) & View 3 (Homepage) về sự thay đổi của Product đó.

d. Web Application Architecture (Kiến trúc ứng dụng Web)

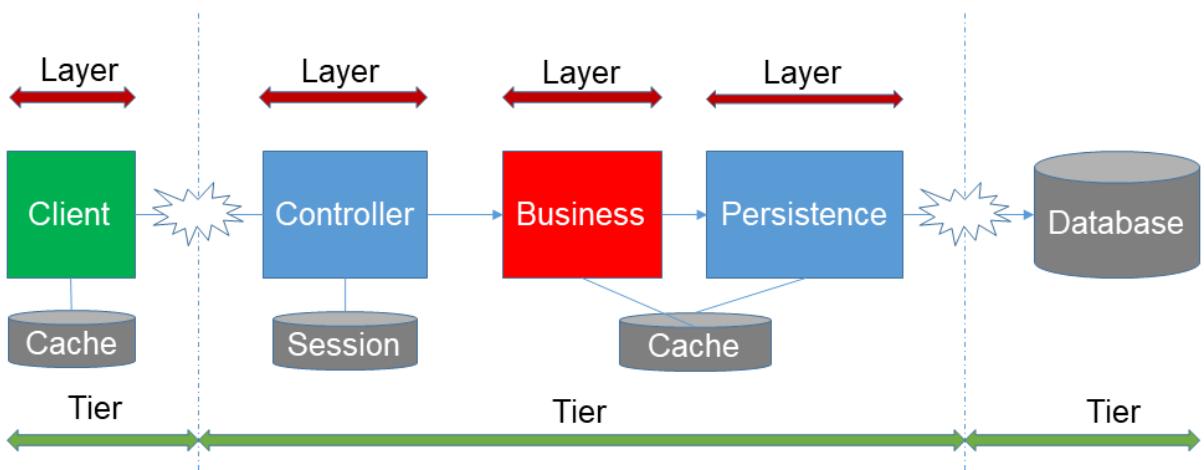
Ứng dụng Web cũng dựa trên Client-Server Model trong đó Web Browser là chương trình Client và Web Server là chương trình Server.

Ứng dụng Web được sử dụng nhiều bởi tính tiện dụng của nó so với ứng dụng Desktop ở chỗ Web Browser luôn có sẵn trong các OS (MS Windows, Linux, Mac OS) trong khi để chạy được chương trình Client ở dạng Desktop phải cài thêm công cụ hỗ trợ (như JRE cho Java Client, .Net Framework cho C# Client...).

Chương trình Web Server thường được phân thành nhiều lớp (Layer) để dễ quản lý, bảo trì hoặc thay đổi công nghệ về sau. Có 3 lớp phổ biến sau:

1. **Controller Layer** (Lớp điều khiển): có nhiệm vụ nhận yêu cầu Web (**Web Request**) từ Web Browser, xử lý các thông số rồi gọi Business Layer xử lý nghiệp vụ.
2. **Business Layer** (Lớp nghiệp vụ): có nhiệm vụ nhận yêu cầu từ Controller Layer để xử lý các nghiệp vụ của hệ thống, trong quá trình xử lý có thể gọi Persistence Layer để truy xuất dữ liệu từ DB hoặc cập nhật dữ liệu xuống DB.
3. **Persistence Layer** (Lớp lưu trữ): có nhiệm vụ nhận yêu cầu từ Business Layer để đọc/ghi dữ liệu lên/xuống DB.

- **Layer:** Logical Separation
- **Tier :** Physical Separation



Sau này nếu có nhu cầu:

- Thay đổi công nghệ Web (ví dụ chuyển từ Jersey sang Spring REST) thì chỉ sửa Code ở Controller Layer, không ảnh hưởng Code ở 2 Layer còn lại.
- Thay đổi công nghệ DB (ví dụ chuyển từ MS SQL sang MySQL) thì chỉ sửa Code ở Persistence Layer, không ảnh hưởng Code ở 2 Layer còn lại.
- Thay đổi nghiệp vụ của hệ thống thì chỉ sửa Code ở Business Layer, không ảnh hưởng Code ở 2 Layer còn lại.

Nếu không phân chia theo Layer mà viết Code chung cho cả 3 Layer, khi phát sinh nhu cầu thay đổi công nghệ Web/DB/nghiệp vụ thì sẽ mất rất nhiều thời gian và công sức để đảm bảo phần sửa Code của mảng này không ảnh hưởng đến Code đang tồn tại của các mảng khác.

e. Logical Separation (Layer) vs Physical Separation (Tier)

Một hệ thống IT có thể được phân chia theo luận lý (**Logical Separation**, gọi là **Layer**) hoặc vật lý (**Physical Separation**, gọi là **Tier**), chúng ta phải phân biệt rõ 2 khái niệm này:

- **Logical Separation:** các Layer cùng nằm trong 1 Source Code, chạy trong cùng 1 vùng nhớ (**Memory Space**).
- **Physical Separation:** các Tier chạy trên các Memory Space khác nhau, có thể trên cùng 1 CPT hoặc nhiều CPT, ví dụ:
 - Web Server Tier (chứa 3 Layers: Controller, Service, Persistence) chạy trên 1 CPT hoặc 1 Virtual Machine trên Cloud.
 - Database Server Tier chạy trên 1 Virtual Machine khác.
 - Web Browser Tier chạy trên Laptop của các User.

f. Multiple Page Application – MPA (Ứng dụng đa trang)

MPA đề cập đến ứng dụng Web có các đặc điểm sau:

- Ứng dụng có nhiều trang Web. Mỗi trang Web được xác định bằng 1 URL.
- Khi User truy cập 1 URL, nội dung trang Web tương ứng, bao gồm nội dung tĩnh (**Static Content**) như hình ảnh, chữ, font và nội dung động (**Dynamic Content**) như dữ liệu sản phẩm, đơn hàng sẽ được sinh ra hoàn toàn ở Web Server. Ví dụ các công nghệ tạo ra trang Web động là CGI, JSP, ASP.Net, PHP...
- Trong thời gian chờ sinh ra trang Web, User chỉ thấy UI trắng trong Web Browser, và nếu lâu quá thì Web Browser sẽ tự ngưng và hiện ra thông báo lỗi. Những ứng dụng nặng về dữ liệu và/hoặc những nơi mà mạng chậm thì việc này xảy ra như cơm bữa làm ảnh hưởng trải nghiệm người dùng. Ngoài ra việc mất thời gian để lấy về các trang Web khiến cho việc tương tác trên UI của User không được mượt như là họ tương tác trên Desktop Application vốn chạy cục bộ trên máy User -> đây là khuyết điểm lớn của MPA.

g. Server-side MVC Pattern (Mẫu MVC phía Server)

Từ lúc Web Application càng trở nên phổ biến hơn Desktop Application thì MVC Pattern cũng được sử dụng rộng rãi vào MPA với cả 3 thành phần đều diễn ra ở Web Server:

1. **Controller:** là Controller Layer, nhận Web Request, gọi Business Layer để xử lý nghiệp vụ rồi nhận lại dữ liệu.
2. **Model:** chứa dữ liệu, trả ra từ Business Layer, mà Controller nhận được để đẩy qua View.
3. **View:** là trang Web sẽ nạp dữ liệu từ Model để trả về cho Web Browser hiển thị cho User.

h. Single Page Application – SPA (Ứng dụng đơn trang)

SPA ra đời để giải quyết khuyết điểm của MPA, với các đặc điểm sau:

- Ứng dụng chỉ có duy nhất 1 trang Web được xác định bằng 1 URL gốc.
- Khi User truy cập URL này, nội dung trang Web, chỉ bao gồm Static Content, được Web Browser lấy về hiển thị cho User liền (có thể lần đầu mất thời gian chút nhưng những lần sau vô sẽ nhanh hơn nhiều do Web Browser đã lưu lại cấu trúc trang Web, chỉ cập nhật những thành phần bị thay đổi). Dynamic Content sẽ được lấy về ngay sau đó. Lúc này User không thấy UI trắng nữa mà thấy được bộ khung UI chưa có dữ liệu. Khi nào Web Browser lấy xong Dynamic Content sẽ hiển thị lên sau.

- Khi User tương tác với các phần UI khác nhau trong trang Web, URL sẽ thay đổi tương ứng. Tuy nhiên không có yêu cầu lấy trang Web nào gửi về phía Web Server như MPA (vì SPA chỉ có duy nhất 1 trang Web mà thôi), thay vào đó việc tạo ra UI khác và chuyển UI được xử lý ở Web Browser dùng PL Javascript sẽ làm cho User sử dụng UI mượt hơn.
- Một nhược điểm của SPA là các nút “Go Back”, “Go Forward”, và “Refresh” của Web Browser sẽ không còn hoạt động đúng như trong MPA, vì vậy FE Dev phải tự xử lý thêm vấn đề này.

i. System Integration (Tích hợp hệ thống)

System Integration đề cập đến việc kết nối 2 Module của 1 hệ thống IT hoặc 2 hệ thống IT với nhau một cách đồng bộ (**Synchronous Integration**) hoặc bất đồng bộ (**Asynchronous Integration**) thông qua các cách sau:

1. **REST Application Programming Interface – API** (hoặc cũ hơn là **Remote Procedure Call – RPC**): đây là phương pháp Synchronous Integration.
2. **Messaging Queue** (Hàng đợi chia thông điệp): đây là phương pháp Asynchronous Integration.
3. **File Sharing** (Chia sẻ File): đây là phương pháp Asynchronous Integration.
4. **Database Sharing** (Chia sẻ DB): đây là phương pháp Asynchronous Integration.

Trong các cách này thì cách 1 & 2 được sử dụng nhiều nhất, tùy loại ứng dụng mà cách 1 hoặc 2 sẽ được chọn:

- Nếu ứng dụng cần sự đồng bộ thì cách 1 được chọn. Ví dụ 1 ứng dụng A về thị trường chứng khoán (Stock Market) cần lấy giá chứng khoán từ 1 hệ thống B, khi A gọi B thì B phải trả kết quả về cho A liền (đồng bộ).
- Nếu ứng dụng cần sự bất đồng bộ thì cách 2 được chọn. Ví dụ 1 ứng dụng X trong mảng Digital Marketing, khi User bấm vào 1 liên kết ngắn (**Short Link**) như goo.gl/abcxyz được chia sẻ trên 1 mạng xã hội, Module 1 trong X sẽ chuyển tiếp User đến trang Web mà Short Link này chỉ đến, đồng thời sẽ gửi 1 Message đến Module 2 trong X để tăng số lần Short Link này được bấm lên 1 đơn vị, lúc Module 1 gửi Message thì Module 2 không cần thiết phải đang chạy để xử lý liền, có thể xử lý sau (bất đồng bộ).

j. Monolithic Application vs Micro-services Architecture

Hồi trước các ứng dụng thuộc dạng Monolithic, có nghĩa là các thành phần, Module cấu tạo nên ứng dụng phải được xây dựng và triển khai cùng lúc thì User mới sử dụng được.

Dạo gần đây 1 dạng SWAR mới ra đời và ngày càng được sử dụng nhiều, đó là Micro-services Architecture. Ứng dụng được xây dựng theo Micro-services Architecture thì được chia nhỏ thành các Micro-service được phát triển, triển khai và chạy độc lập với nhau, giao tiếp với nhau thông qua các REST API.

Mỗi loại SWAR đều có ưu/nhược riêng, chọn SWAR nào tùy thuộc vào nhiều yếu tố.

Monolithic Architecture	Micro-service Architecture
Pros: <ol style="list-style-type: none"> Ít gặp các vấn đề liên quan đến toàn hệ thống (Cross-cutting Concerns) và chỉ nằm trong 1 ứng dụng nên cũng dễ xử lý. Dễ debug và test toàn hệ thống. Dễ triển khai. Phát triển đơn giản. 	Pros: <ol style="list-style-type: none"> Từng micro-service nhỏ nên dễ hiểu. Các micro-service độc lập nên dễ thay đổi và bị bug ở một vài micro-service không ảnh hưởng toàn bộ ứng dụng. Co giãn trên từng micro-service đơn giản. Có thể sử dụng nhiều công nghệ chung.
Cons: <ol style="list-style-type: none"> Khi ứng dụng càng lớn thì sẽ càng phức tạp, khó hiểu và khó quản lý. Khó khăn hơn để thay đổi toàn hệ thống dẫn đến việc phát triển sẽ kéo dài hơn. Khó mà co giãn các Module riêng rẽ, đôi khi phải co giãn toàn bộ hệ thống. Khó đưa một công nghệ mới vào, đôi khi phải viết lại toàn bộ ứng dụng. 	Cons: <ol style="list-style-type: none"> Phức tạp trong việc quản lý các mối quan hệ giữa các micro-service. Khó debug và test toàn hệ thống. Triển khai nhiều micro-service phức tạp. Phát triển nhiều micro-service phức tạp hơn.

k. Self-Experience

- Viết 1 Program theo Client-Server Model đơn giản
 - Client kết nối đến Server.
 - Client gửi message đến Server thì Server trả lại message, đây gọi là **Echo Server**.
- Viết 1 Desktop Application đơn giản:
 - Xây dựng UI cho phép tạo Product, tìm kiếm Product, chỉnh sửa Product, và xóa Product.
 - Xây dựng Database Server để lưu trữ Product.

3. Viết 1 MPA dùng MVC Pattern đơn giản:

- Xây dựng các trang Web cho phép tạo Product, tìm kiếm Product, chỉnh sửa Product, và xóa Product.
- Xây dựng Web Server gồm 3 Layers.
- Xây dựng Database Server để lưu trữ Product.

4. Viết 1 SPA đơn giản:

- Xây dựng 1 trang Web duy nhất có các UI cho phép tạo Product, tìm kiếm Product, chỉnh sửa Product, và xóa Product (có thể dùng jQuery Lib).
- Xây dựng Web Server gồm 3 Layers trả về dữ liệu cho UI.
- Xây dựng Database Server để lưu trữ Product.

2) Backend Programming – BEP (Lập trình phía Backend)

Ngày nay SPA đã hầu như thay thế MPA thì thế giới lập trình cũng tách thành 2 mảng là lập trình phía Backend (**Backend Programming – BEP**) và lập trình phía Frontend (**Frontend Programming – FEP**).

Trong chương này chúng ta sẽ tìm hiểu những khái niệm quan trọng cũng như các mục mà BE Dev cần phải nắm. Phần FEP sẽ được trình bày trong chương tiếp theo.

a. Application Programming Interface – API (Giao tiếp lập trình ứng dụng)

API là 1 giao thức giao tiếp (**Communication Protocol**) để kết nối các phần khác nhau của một Program bằng phương pháp trừu tượng hóa (**Abstraction**) với mục đích là đơn giản hóa việc hiện thực và bảo trì các thành phần đó, có nghĩa là ai xài API thì chỉ quan tâm đến tính năng của nó mà không cần quan tâm nó được hiện thực bởi công nghệ gì.

Sau này API được mở rộng ra để kết nối các hệ thống IT khác nhau, có thể của các doanh nghiệp khác nhau.

Ở mức độ doanh nghiệp, API được cung cấp có các dạng sau:

- **Private:** API chỉ được sử dụng trong nội bộ doanh nghiệp.
- **Partner:** Chỉ các doanh nghiệp cộng tác mới được sử dụng API.
- **Public:** Bất kỳ doanh nghiệp hay cá nhân nào cũng có thể sử dụng API.

b. REpresentational State Transfer – REST Architeture (Kiến trúc REST)

Trước kia, một dạng cung cấp API phổ biến là Web Service được hiện thực bằng giao thức **SOAP** (Simple Object Access Protocol) trong đó Web Browser gửi 1 yêu cầu theo định dạng XML (**eXtension Markup Language**), Web Server xử lý yêu cầu rồi trả về kết quả cũng theo định dạng XML.

Nói chung SOAP Web Service khá là rắc rối và phức tạp với cả núi XML. Do đó, một kiến trúc mới ra đời dần thay thế SOAP, đó là REST.

REST định nghĩa một tập hợp các ràng buộc trong việc tạo ra Web Service làm API. Web Service tuân thủ các qui tắc của REST được gọi là RESTful Web Service, từ gọi thông dụng hơn là RESTful API, gọn hơn là REST API.

REST API được sử dụng phổ biến nhất với giao thức HTTP và định dạng dữ liệu JSON:

- Web Browser gọi REST API với JSON data
- REST API xử lý yêu cầu rồi trả về JSON data

c. JSON Processing (Xử lý JSON)

Với việc REST trở thành chuẩn thông dụng cho API thì BEP sẽ dụng đến vấn đề xử lý JSON (**JSON Processing**).

JSON (JavaScript Object Notation) là một định dạng dữ liệu theo kiểu “key-value” đơn giản và gọn nhẹ hơn XML nên được sử dụng để thay thế XML trong việc truyền dữ liệu qua mạng và xử lý dữ liệu.

Ở đây chúng ta cần biết đến 2 khái niệm:

- **Problem Space** (Không gian bài toán): là nơi chứa các đối tượng trong lĩnh vực (**Domain Object**) mà SW đang hoạt động, ví dụ: trong lĩnh vực eCommerce thì các Object sẽ là Product, Order, Order Line, Shipment, Customer..., trong lĩnh vực giáo dục thì các Object sẽ là University, Student, Lecturer...
- **Solution Space** (Không gian giải pháp): là nơi chứa các Object của công nghệ được dùng để xây dựng SW như Javascript, Java, C#.

JSON Processing bao gồm các công việc sau:

- Chương trình Frontend gọi REST API truyền JSON data dạng chuỗi (String) thu thập được từ Problem Space.
- Controller Layer nhận yêu cầu từ Web Browser, chuyển JSON data dạng String sang dạng đối tượng (Object) như Java Object, C# Object trong Solution Space. Sau khi gọi Business Layer xử lý nghiệp vụ xong, kết quả trả ra cũng thuộc dạng Object, Controller Layer chuyển kết quả này sang JSON data dạng String (có thể thêm một số thuộc tính phụ khi cần) trả về cho chương trình Frontend.
- Chương trình Frontend chuyển JSON data dạng String nhận được từ REST API sang dạng Javascript Object trong Solution Space để xử lý tiếp.

d. Object Mapping (Ánh xạ đối tượng)

Khái niệm này liên quan đến việc chuyển đổi giữa các dạng Object bên Web Server.

Theo mô hình 3 Layer đã đề cập ở trên thì mỗi Layer sẽ nhận vào và trả ra các loại Object khác nhau:

- Controller Layer nhận vào dạng Object gọi là View Model, có thể hiểu là dữ liệu thu thập được từ UI, trả về JSON data dạng String cho Web Browser.
- Business Layer nhận vào View Model từ Controller Layer, có thể ánh xạ sang **Transient Domain Object – TDO** hoặc **Detached Domain Object – DDO**, sau đó gọi Persistence Layer truyền TDO/DDO vào rồi nhận về Persistence Domain Object – PDO, cuối cùng ánh xạ từ PDO sang **Data Transfer Object – DTO** để trả DTO về cho Controller Layer.
- Persistence Layer nhận vào TDO/DDO, xử lý xong trả về PDO cho Business Layer.

e. Interceptor (Bộ chặn) & Web Filter (Bộ lọc Web)

Interceptor hiểu nôm na là cách thức can thiệp 1 Procedure Call để xử lý thêm việc gì đó trước khi để Procedure Call chạy như thông thường.

Interceptor thường được sử dụng trong trường hợp Procedure đã tồn tại và việc thay đổi nội dung Procedure là khó khăn. Ví dụ: một Procedure thực hiện 1 nghiệp vụ nào đó đang chạy bình thường, giờ chúng ta muốn thêm việc kiểm tra xem User đang đăng nhập có được phân quyền để thực hiện gọi Procedure này hay không mà không can thiệp vào nội dung đã có của Procedure -> sử dụng Interceptor.

Web Filter là một dạng ứng dụng của Interceptor trong môi trường Web Server. Trước khi yêu cầu từ Web Browser đi đến Controller Layer, nó sẽ đi qua 1 số Web Filter để xử lý những việc gì đó.

Các Web Filter nối tiếp nhau tạo thành 1 chuỗi lọc (**Filter Chain**). Mỗi Web Filter trong Filter Chain nhận vào Web Request để xử lý, nếu mọi thứ ổn thỏa thì Web Filter này sẽ chuyển tiếp Web Request qua Web Filter tiếp theo, còn nếu có điều kiện gì đó mà Web Request không thỏa mãn thì Web Filter sẽ ngắt và trả kết quả lỗi về cho Web Browser.

Ví dụ: Spring Security FW sử dụng rất nhiều Web Filter để hiện thực phần bảo mật (bao gồm User Authentication & User Authorization) cho SW.

f. Inversion of Control – IoC (Đảo ngược điều khiển)

Thông thường 1 Program sẽ chạy từ đầu đến cuối. Trong khi chạy thì Program có thể gọi các Procedure khác của Lib/FW, luồng thực thi (**Execution Flow**) được Program điều khiển.

Đảo ngược điều khiển (**Inversion of Control – IoC**) là 1 nguyên tắc lập trình (**Programming Principle**) cho phép đôi lúc Execution Flow được điều khiển bởi Lib/FW giúp cho việc phát triển SW nhanh hơn và linh hoạt hơn.

Hai hiện thực của IoC được sử dụng phổ biến là lập trình dựa vào sự kiện (**Event-Driven Programming**) và Tiêm sự phụ thuộc (**Dependency Injection – DI**) mà chúng ta sẽ tìm hiểu ngay sau đây.

g. Event-Driven Programming (Lập trình dựa vào sự kiện)

Event-Driven Programming là cách thức lập trình trong đó Program chạy đến 1 đoạn nào đó thì sẽ chờ sự kiện để xử lý. Các sự kiện có thể là hành động của User, thông điệp được gửi từ các tiểu tiến trình (**Thread**) hoặc các Program khác.

Program đăng ký các bộ xử lý sự kiện (**Event Handler**) cho các sự kiện (thường được cung cấp bởi các Lib/FW) để khi 1 sự kiện xảy ra thì Lib/FW sẽ gọi Event Handler của Program thực thi xử lý sự kiện đó.

h. Dependency Injection – DI (Tiêm sự phụ thuộc)

Trong OOP, Dependency Injection là 1 kỹ thuật để cung cấp 1 Object B mà 1 Object A cần để sử dụng (hay nói cách khác là Object A phụ thuộc vào Object B).

Ví dụ: Class “GenericDao” (dùng để tương tác với DB) phụ thuộc vào Class “MySQLConnector” để kết nối với DB MySQL.

```
class GenericDao {  
    MySQLConnector mySQLConnector = new MySQLConnector();  
    ...  
}
```

Nếu lập trình như vậy thì sau này nếu có đổi qua DB khác như PostgreSQL thì Class “GenericDao” phải bị sửa lại: **PostgreSQLConnector postgreSQLConnector = new PostgreSQLConnector();** -> biên dịch lại Program rồi triển khai lại.

Nếu sử dụng DI thì sau này có đổi qua DB khác thì Class “GenericDao” sẽ không bị sửa lại mà chỉ cần chỉnh cấu hình là được:

```
class GenericDao {
    @Injected
    DBConnector dbConnector;
    ...
}
```

Kí hiệu chú thích (**Annotation**) `@Injected` báo cho DI Container (được cung cấp bởi 1 FW nào đó, chứa tất cả Object trong hệ thống) biết là Object thuộc Class “GenericDao” khi chạy sẽ cần Object `dbConnector` thuộc Class “DBConnector” để DI Container sẽ tiêm vào Object `dbConnector` phù hợp (MySQL hoặc PostgreSQL tùy thuộc vào cấu hình cho Class DBConnector).

DI ngoài việc tạo sự linh hoạt như trên còn giúp cho việc tái sử dụng lại Object mà không phải khởi tạo, ví dụ: Class “SpecificDao” cũng cần Object `dbConnector`:

```
class SpecificDao {
    @Injected
    DBConnector dbConnector;
    ...
}
```

i. User Authentication (Định danh người dùng)

Tính năng User Authentication đảm bảo chỉ có những ai có tài khoản trong hệ thống mới được đăng nhập vào hệ thống để sử dụng các chức năng hệ thống cung cấp.

Thứ ban đầu, việc hiện thực User Authentication sử dụng 2 thông tin cơ bản là tên người dùng (**Username**) và mật khẩu (**Password**). Sau này thì thư điện tử (Email) hoặc điện thoại (Phone) thay thế Username.

Để sử dụng được 1 hệ thống A nào đó, một người phải đăng ký 1 tài khoản người dùng (User Account) trên hệ thống A với 2 thông tin ở trên. Tuy nhiên, nếu người này sử dụng vài chục hệ thống thì phải tạo vài chục User Account và phải nhớ/lưu tất cả các thông tin đăng nhập.

Với những hệ thống quan trọng như ngân hàng, bệnh viện thì User sẽ chấp nhận chuyện tạo tài khoản, còn những hệ thống ít hoặc không quan trọng như những công cụ online cho phép tạo tài liệu, hình ảnh... thì User sẽ ngại tạo tài khoản và ít sử dụng.

Thế là các hệ thống chưa/ít nổi tiếng muốn thu hút User tham gia sử dụng thì phải cung cấp cơ chế User Authentication sao cho họ không cần phải tạo tài khoản trên đó?

Một cơ chế User Authentication mới đã ra đời, đó là OAuth (**Open Authentication**) cho phép User đăng nhập vào 1 hệ thống mà không phải tạo tài khoản trên hệ thống đó, thay vào đó là sử dụng 1 tài khoản của những hệ thống nổi tiếng như Google, Facebook, Microsoft... mà hệ thống này hỗ trợ.

j. User Authorization (Phân quyền người dùng)

Tính năng User Authentication ở trên giúp định danh một tài khoản phải tồn tại trong hệ thống mới được đăng nhập vào hệ thống.

Sau khi đăng nhập vào được hệ thống, tính năng User Authorization giúp xác định những chức năng nào của hệ thống mà tài khoản đang đăng nhập được sử dụng. Để làm được việc này thì tài khoản phải có thêm thông tin phân quyền có thể là vai trò phân quyền (**Authorization Role**) và/hoặc cho phép phân quyền (**Authorization Permission**) tùy thuộc vào cách thiết kế mô hình bảo mật hệ thống.

Thông thường có 2 mức phân quyền:

- **Method-Level Authorization** (Phân quyền mức phương thức): các Method được chú thích vai trò nào được gọi, chỉ tài khoản đang đăng nhập nào có vai trò nằm trong danh sách các vai trò cho phép của 1 Method mới được gọi Method đó.
- **Data-Level Authorization** (Phân quyền mức dữ liệu): Ví dụ: trong 1 hệ thống eCommerce, tài khoản với vai trò “ADMIN” khi đăng nhập vào sẽ truy xuất được toàn bộ Customer, Order có trong hệ thống trong khi tài khoản với vai trò “CUSTOMER” khi đăng nhập vào không được truy xuất các Customer khác, những Order của các Customer khác trong hệ thống, mà chỉ có thể truy xuất được các Order của chính họ.

k. System Logging (Lưu vết hệ thống)

Tính năng System Logging giúp lưu lại quá trình hệ thống chạy cùng với những thông tin khi hệ thống xử lý các yêu cầu để từ đó giúp chúng ta nghiên cứu xử lý sự cố khi phát sinh.

I. Data Validation (Xác nhận dữ liệu)

Khái niệm Data Validation chỉ đến việc kiểm tra xem dữ liệu đưa vào hệ thống có đúng như yêu cầu của hệ thống hay không, điều này giúp hạn chế những lỗi liên quan đến sai dữ liệu đầu vào (**Input Data**).

Các tiêu chí để kiểm tra dữ liệu rơi vào các dạng sau:

- **Required:** dữ liệu bắt buộc phải có, ví dụ: khi tạo tài khoản mới thì email và password bắt buộc phải cung cấp.
- **Format:** dữ liệu phải thỏa định dạng nào đó, ví dụ: email phải đúng định dạng email, phone phải chỉ chứa số, không được chứa chữ.
- **Length:** dữ liệu phải thỏa chiều dài nào đó, ví dụ: username phải dài ít nhất 6 ký tự, password phải dài ít nhất 8 ký tự.
- Các ràng buộc khác tùy thuộc nghiệp vụ.

m. Business Transaction (Giao dịch nghiệp vụ)

Trong phần kiến thức Database chúng ta đã biết về Database Transaction là giao dịch xảy ra bên dưới DB.

Trong Business Layer, có thêm 1 khái niệm giao dịch nữa là giao dịch nghiệp vụ (**Business Transaction**) là giao dịch ở mức cao hơn Giao dịch DB. Về mặt tính chất thì nó cũng có tính All-or-nothing như Database Transaction.

Các Business Transaction được hiện thực dưới dạng các Method có thể gọi lân nhau để sử dụng lại các nghiệp vụ sẵn có. Mỗi Transactional Method này sẽ luôn chạy trong 1 Transaction có thể được tạo mới trước khi vào Method này hoặc sử dụng Transaction đang có của Method gọi nó (tùy theo cấu hình Transaction cho nó).

n. Object Relational Mapping – ORM (Ánh xạ quan hệ & đối tượng)

ORM là kỹ thuật chuyển đổi thị đối tượng (**Object Graph**) sang các Table trong DB quan hệ. Với ORM, SD không cần phải viết lệnh SQL để ghi/đọc dữ liệu vốn rót thời gian và dễ sai sót, vì vậy Code sẽ ngắn hơn. Thay vào đó, SD chỉ cần đặc tả cách ánh xạ từ Object Graph sang các Table trong DB quan hệ, FW hiện thực ORM sẽ sinh ra các lệnh SQL rồi thực thi.

Các vấn đề cần lưu tâm khi thực hiện ánh xạ:

- **Inheritance Mapping:** có 3 cách ánh xạ sự thừa kế, chọn cách nào tùy thuộc vào yêu cầu cụ thể của hệ thống.
- **Data Type:** kiểu dữ liệu trên Object & DB khác nhau, phải chọn sao cho phù hợp tùy thuộc vào yêu cầu cụ thể của hệ thống.
- **Relationship Mapping:** không nên sử dụng quan hệ nhiều-nhiều trên Object vì ánh xạ xuống Table sẽ rất phức tạp, nên tách quan hệ nhiều-nhiều thành 2 quan hệ 1-nhiều rồi mới thực hiện ánh xạ.
- **Lazy Loading vs Eager Fetching:** cân nhắc giữa 2 tình huống là khi nào thì lấy dữ liệu khi có yêu cầu với khi nào lấy trước dữ liệu mà chưa cần có yêu cầu.
- **Cascading:** phải kiểm soát được việc tạo mới/thay đổi/xóa 1 Object sẽ ảnh hưởng dây chuyền đến các Object quan hệ với nó.
- **Data Navigation:** với mỗi quan hệ 1-1 hay 1-nhiều thì các Table trong DB có thể truy xuất dữ liệu 2 chiều (Bi-direction) trong khi Object thì có thể truy xuất 1 chiều (Uni-direction) hoặc 2 chiều tùy trường hợp.
- **“N+1” Select Problem:** đây là vấn đề rất phổ biến khi ánh xạ quan hệ 1-nhiều ảnh hưởng đến System Performance, cần lưu ý tránh trong những trường hợp có thể xảy ra.

o. Data Search/Sorting/Paging (Tìm kiếm/Sắp xếp/Phân trang dữ liệu)

Khi SW được sử dụng một thời gian thì sẽ có nhiều dữ liệu, tính năng Data Search/ Sorting/ Paging giúp User có thể tìm kiếm dữ liệu tiện lợi hơn và nhanh hơn, vì thế tính năng này tồn tại hầu hết trong mọi SW.

p. Error Handling (Xử lý lỗi)

Bất cứ SW nào cũng chắc chắn sẽ có lỗi, không sớm thì muộn. Những lỗi nhỏ có thể làm trải nghiệm của User không tốt, hoặc làm một số tính năng không còn chạy hoặc chạy sai, và những lỗi nghiêm trọng có thể làm hệ thống ngưng hoạt động.

Dù lỗi nhỏ hay lớn thì đều phải được xử lý lỗi (**Error Handling**) một cách rõ ràng nhất có thể để hệ thống chạy ổn định và lưu vết đầy đủ để Troubleshooting các sự cố sau này.

q. Concurrency Handling (Xử lý đồng thời)

Concurrency Handling liên quan đến việc xử lý các tài nguyên được truy cập đồng thời có thể dẫn đến các kết quả ngoài mong đợi.

Các tài nguyên có thể là:

- 1 Property của 1 Object
- 1 đoạn Code trong 1 Method của 1 Object
- 1 dòng dữ liệu trong 1 Table trong DB

Ví dụ về kết quả sai lệch do truy cập tài nguyên đồng thời:

- Có 1 URL được chia sẻ trên internet, mỗi lần URL được bấm thì biến đếm của nó trong DB sẽ tăng lên 1 đơn vị.
- Giả sử biến đếm của URL trong DB đang là 10, URL được bấm 1 phát, Thread A xử lý hành động này đọc biến đếm của URL lên bộ nhớ để tăng lên 1 là 11.
- Trong khi Thread A chưa kịp lưu biến đếm (đang là 11) xuống DB thì URL được bấm 2 phát và Thread B, C trên máy khác (chạy nhanh hơn Thread A) đã đọc biến đếm lên, tăng lên 11, 12 rồi lưu xuống. Lúc này Thread A mới lưu biến đếm xuống DB đè lên giá trị 12 thành 11 -> kết quả cuối cùng bị sai (phải là 13 thay vì 11), nguyên nhân là trong thời gian Thread A đang xử lý biến đếm thì nó đã bị thay đổi bởi các Thread khác.

Có 2 cách để thực hiện Concurrency Handling:

- **Pessimistic Locking** (Khóa bi quan): Như ví dụ trên thì khi Thread A truy cập biến đếm thì sẽ khóa biến đếm này lại, không cho bất kỳ Thread nào truy cập, khi nào Thread A xử lý xong sẽ mở khóa. Cách này đảm bảo dữ liệu không bị sai nhưng sẽ làm hệ thống chậm đi do các Thread khác phải chờ Thread. Tính bi quan của cách này là sợ sai sót xảy ra nên chặn ngay từ đầu.
- **Optimistic Locking** (Khóa lạc quan): Khi Thread A truy cập biến đếm thì biến đếm này sẽ không bị khóa, các Thread khác vẫn truy cập được, tuy nhiên khi bắt đầu Thread nào cập nhật biến đếm này thì sẽ bị kiểm tra coi dữ liệu có khả năng bị sai không (cách thông dụng là dùng Version để kiểm tra sự thay đổi). Tính lạc quan của cách này là nghĩ sai sót sẽ không xảy ra, tuy nhiên nếu lỡ xảy ra thì có cách xử lý ổn thỏa. Trong ví dụ trên khi Thread A cập nhật biến đếm xuống DB, phải kiểm tra Version của biến đếm Thread A đang giữ có bằng với Version hiện tại của biến đếm trong DB hay không? Nếu không bằng thì bắt Thread A phải đọc biến đếm lại, nếu bằng thì cho cập nhật.

r. Asynchronous Processing (Xử lý bất đồng bộ)

Trong BEP, các trường hợp phổ biến cần thực hiện Asynchronous Processing sau:

- Khi User thực hiện 1 hành động gì đó, sau khi hệ thống xử lý xong nghiệp vụ thì cần phải tạo thông báo (Notification) và gửi email cho những User liên quan, 2 việc này sẽ tốn thời gian và User không cần quan tâm đến 2 việc này có thành công hay không. Cho nên hệ thống sẽ trả kết quả về cho User ngay sau khi xong nghiệp vụ trong Thread đang phục vụ User để User không phải chờ, còn 2 việc này sẽ được thực thi ngầm bên dưới hệ thống thông qua 1 Thread khác.
- Khi User thực hiện 1 hành động gì đó, sau khi hệ thống xử lý xong nghiệp vụ thì cần báo cho Module khác hoặc hệ thống khác mà Module /hệ thống khác không phải lúc nào cũng sẵn sàng phục vụ, hệ thống sẽ gửi thông điệp sử dụng phương pháp Messaging Queue trong phần System Integration đã đề cập ở trên.

s. Resource Pooling Configuration (Threads, DB Connections...)

Khái niệm Resource Pooling chỉ đến việc quản lý tài nguyên trong hò (Pool) nhằm tăng hiệu quả của hệ thống và hạn chế lãng phí tài nguyên.

Trong ngữ cảnh của Web Server, mỗi lần Web Request gửi đến Web Server, cần 1 Thread để xử lý. Chi phí & thời gian để tạo ra 1 Thread là đáng kể, cho nên thay vì cứ 1 Web Request đến sẽ tạo ra 1 Thread xử lý thì sẽ tạo sẵn 1 lượng Thread nào đó bỏ vào 1 cái hò. Các Thread này sẽ thay nhau xử lý các Web Request. Khi lượng Web Request tăng lên quá số lượng Thread thì sẽ tạo thêm Thread xử lý, xong thì sẽ hủy các Thread này.

Trường hợp cần DB Connection kết nối đến DB cũng sẽ xử lý tương tự.

Resource Pool được hiện thực trong thực tế còn có thêm 1 hàng đợi (**Queue**) (hình nằm trong phần System Performance ở dưới). Việc cấu hình cho các Resource Pool sao cho tối ưu nhất tùy thuộc vào tình huống cụ thể của hệ thống ở mỗi thời điểm.

t. Big File Downloading Handling (Xử lý việc tải file lớn)

Trong SW, đôi khi User muốn trích xuất dữ liệu như đơn hàng trong tháng từ DB ra File để làm báo cáo. Nếu lượng dữ liệu nhỏ, BE sẽ đọc từ DB lên rồi bỏ vào File cho User tải về.

Tuy nhiên nếu lượng dữ liệu đủ lớn, quá trình đọc DB lên rồi bỏ vào File sẽ mất rất nhiều thời gian làm User phải chờ rất lâu (có thể dẫn đến Web Browser ngắt kết nối), lúc này giải pháp khả dĩ nhất là thực hiện Asynchronous Processing:

- Khi User yêu cầu trích xuất dữ liệu, hệ thống trả về liền thông báo cho User biết là hệ thống đang xử lý việc xuất dữ liệu.
- Hệ thống tạo 1 Thread thực hiện việc đọc dữ liệu từ DB rồi bỏ vào File, sau đó đẩy File lên 1 nơi lưu trữ nào đó, lấy URL trả về lưu vô DB.

u. Data Caching (Lưu dữ liệu tạm thời)

Với những dữ liệu không thay đổi hoặc rất ít thay đổi mà được truy xuất thường xuyên, việc đọc từ DB liên tục sẽ làm chậm hệ thống.

Khái niệm Data Caching giúp lưu những loại dữ liệu dạng này vào một nơi mà việc đọc dữ liệu sẽ nhanh hơn đọc từ DB như lưu trong bộ nhớ hoặc Redis (một NoSQL DB).

Khi sử dụng Data Caching cần lưu ý xử lý việc đồng bộ loại dữ liệu rất ít thay đổi giữa Cache và DB để cho Cache chứa dữ liệu mới nhất có thể.

v. Internalization – I18n (Đa ngôn ngữ)

Tính năng I18n giúp cho hệ thống hỗ trợ nhiều ngôn ngữ như tiếng Anh, tiếng Việt... để phục vụ cho nhiều loại User sử dụng các ngôn ngữ khác nhau hoặc ở các nước khác nhau.

w. External Integration: SMS, Payment, Mass Emails, AWS...

Khi xử lý yêu cầu của User, hệ thống thường sử dụng một số tính năng được cung cấp bởi bên thứ 3 (**3rd-party Provider**) như nhắn tin SMS, thanh toán, gửi email hàng loạt... Cho nên việc hệ thống được tích hợp với bên ngoài (**External Integration**) là điều thiết yếu.

Thực hiện External Integration phải lưu ý đến việc Error Handling vì lúc này hệ thống phụ thuộc vào “sức khỏe” của các hệ thống khác nữa.

x. Unit Test (Kiểm tra đơn vị)

Thực hiện Unit Test giúp kiểm soát việc thay đổi hệ thống tốt hơn nhưng sẽ mất rất nhiều thời gian và công sức.

y. Self-Experience

Xây dựng phần REST API cho 1 ứng dụng eCommerce cơ bản (sử dụng những khái niệm đã được trình bày ở trên) gồm các phần sau:

- User đăng nhập hệ thống bằng email & password.
- User có 1 trong 2 Role: ADMIN & CUSTOMER.
- User ADMIN có quyền tạo Admin khác, tìm Admin, sửa Admin, và xóa Admin (không được xóa chính nó).
- User ADMIN có quyền tạo Product, tìm Product, sửa Product, và xóa Product.
- User chưa có tài khoản CUSTOMER thì được đăng ký 1 tài khoản mới.
- User CUSTOMER có quyền tìm Product, chọn mua Product, tạo Order (giả sử trả bằng tiền mặt là Cash On Delivery – COD), tìm Order họ đã mua, xem chi tiết 1 Order của họ.
- User ADMIN có quyền tìm Order, xem chi tiết 1 Order.

Một bộ REST API mẫu cho phần Product:

1. Tạo 1 Product: URL: **/products**, HTTP POST
2. Đọc 1 Product: URL: **/products/{productId}**, HTTP GET
3. Sửa 1 Product: URL: **/products/{productId}**, HTTP PUT
4. Xóa 1 Product: URL: **/products/{productId}**, HTTP DELETE
5. Tìm nhiều Product: URL: **/products/search**, HTTP POST

Sau khi làm xong thì có thể sử dụng công cụ Postman để kiểm thử.

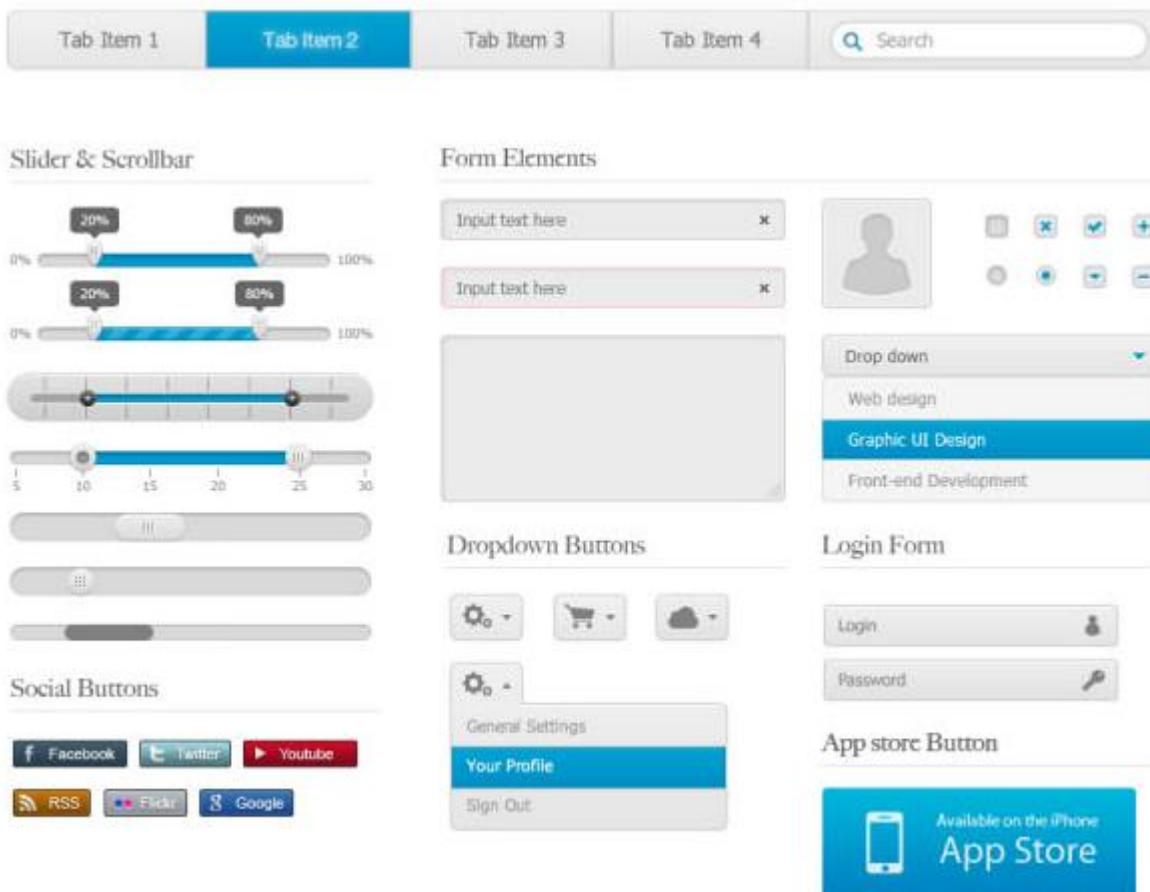
3) Frontend Programming – FEP (Lập trình phía Frontend)

Chương trước chúng ta đã tìm hiểu những khái niệm & tính năng bên BEP, chương này chúng ta sang tìm hiểu bên FEP.

a. User Interface – UI (Giao diện người dùng)

Nhiệm vụ chính của FEP là xây dựng các UI giúp cho User tương tác với SW dễ dàng, thuận tiện và hiệu quả nhất. Mỗi UI được thiết kế để đáp ứng mỗi loại nghiệp vụ đơn giản hay phức tạp nào đó. Dù UI thế nào đi nữa thì cũng bao gồm những phần tử (**UI Element**) cơ bản sau:

- **Label:** đây là phần tử thông dụng nhất được sử dụng cho User biết thông tin trên UI.
- **Text Field:** đây là phần tử thông dụng nhất dùng để nhập dữ liệu trên 1 dòng. Dữ liệu có thể là chữ, số, ký tự đặc biệt, mật khẩu...
- **Text Area:** phần tử này dùng để nhập chuỗi có thể kéo dài trên nhiều dòng.



- **Combo Box / Dropdown List:** phần tử này hiển thị 1 danh sách dữ liệu cho User chọn, User có thể gõ từ khóa trên phần tử này để tìm dữ liệu cần chọn cho nhanh.
- **Check Box:** phần tử này cho phép người dùng đánh dấu Có/Không.
- **Radio Button:** nhóm các phần tử dạng này cung cấp cho User lựa chọn chỉ một trong các giá trị, ví dụ: Nam/Nữ, Độc thân/Kết hôn/Li dị/Góa bụi.

- **Button:** phần tử này cho phép User thực hiện 1 hành động gì đó khi bấm vào.

Từ những phần tử cơ bản, các phần tử phức tạp hơn được tạo ra để giúp User thao tác với UI nhanh và tiện lợi hơn như hình ở trên.

b. Hyper Text Markup Language – HTML

Để xây dựng một trang Web thì chúng ta cần phải dùng 1 ngôn ngữ đó là HTML giúp định nghĩa nội dung trang Web sẽ bao gồm những gì. Một điểm nổi bật của trang Web là có thẻ liên kết với các trang Web khác qua thẻ (**Tag**) `<a>`.

Ví dụ:

```
<html>
  <body>
    <h1>Hello Web World</h1>
    <a href='http://google.com'>Go to Google page</a>
  </body>
</html>
```

c. Cascading Style Sheet – CSS

Ngôn ngữ HTML cho phép chúng ta định nghĩa nội dung trang Web thì ngôn ngữ CSS giúp chúng ta tùy chỉnh hình thức trang Web.

Ví dụ: chỉnh cho tiêu đề h1 ở trên có màu xanh và in nghiêng

```
<html>
  <style>
    h1 {
      color: blue;
      font-style: italic;
    }
  </style>

  <body>
    <h1>Hello Web World</h1>
    <a href='http://google.com'>Go to Google page</a>
  </body>
</html>
```

d. Javascript

Sau khi đã xây dựng được nội dung và hình thức cho trang Web, tiếp theo chúng ta sử dụng PL Javascript để cho User tương tác được với trang Web. Lập trình Javascript cần nắm vững các mục dưới đây:

- function
- function prototype
- event
- closure
- call & apply
- callback, promise, async/await
- timer
- code evaluation
- AJAX

e. Event Handling (Xử lý sự kiện)

Mỗi khi User thực hiện hành động trên 1 UI Element, một sự kiện được sinh ra trên UI Element đó. Các sự kiện phổ biến là:

- User nhập dữ liệu trên 1 Text Field rồi bấm nút Enter -> “onKeyDown” Event.
- User xổ Dropdown List xuống rồi chọn 1 giá trị -> “onChange” Event.
- User chọn/bỏ chọn 1 Check Box -> “onChange” Event.
- User thay đổi chọn lựa trên 1 nhóm Radio Button -> “onChange” Event.
- User bấm vào 1 Button -> “onClick” Event.

Thế thì để User tương tác được với trang Web, chúng ta sẽ viết từng bộ xử lý sự kiện (**Event Handler**) trên mỗi UI Element có khả năng xảy ra sự kiện.

Ví dụ: trong 1 UI tạo 1 Product, sau khi User nhập dữ liệu vào tất cả các UI Element trong mẫu (**Form**) tạo Product và bấm vô Button “Save” -> đoạn Javascript Code chúng ta viết cho Event Handler xử lý sự kiện “onClick” trên Button “Save” sẽ được thực thi để tạo ra 1 Product mới trong hệ thống.

f. Event Bubbling (Chuyển tiếp sự kiện)

Như chúng ta thấy ở trên thì HTML cung cấp cấu trúc lồng nhau để định nghĩa nội dung trang Web, có nghĩa là các UI Element sẽ có quan hệ cha & con. UI Element lớn nhất là <html>.

Javascript sử dụng cấu trúc Object lồng nhau để quản lý các UI Element. Object lớn nhất có tên là “window”, tiếp theo là “document” -> “body” -> các UI Element được định nghĩa bởi người xây dựng trang Web.

Một tính chất đặc biệt của Javascript liên quan đến sự kiện là khi sự kiện xảy ra trên 1 UI Element, ngoài sự kiện được sinh ra trên UI Element đó thì sự kiện này cũng sẽ được chuyển tiếp lên (**Event Bubbling**) UI Element cha -> ông nội -> ... -> “body” -> “document” -> “window”.

Thế thì tính chất đặc biệt này có ý nghĩa gì trong ứng dụng thực tế?

Thông thường khi xây dựng trang Web dựa trên bản thiết kế UI thì chúng ta biết trước UI sẽ có những UI Element nào, là các phần tử tĩnh (**Static Element**), và phải viết Event Handler cho các phần tử có sự kiện xảy ra trên đó, rồi đăng ký các Event Handler vô các sự kiện.

Tuy nhiên có những trang Web có phần tử động (**Dynamic Element**), là những phần tử được sinh ra/xóa đi trong quá trình User tương tác với trang Web, có sự kiện xảy ra trên đó, ví dụ: phần tử TODO (việc cần làm) trong ứng dụng quản lý TODO.

Cứ mỗi Dynamic Element được tạo ra thì phải viết Event Handler rồi đăng ký vô sự kiện, rồi khi nó bị xóa đi thì phải tháo Event Handler ra khỏi sự kiện, nói chung rất là phức tạp và tốn công sức. Giải pháp tốt hơn cho bài toán Dynamic Element này là sử dụng Event Bubbling -> viết 1 Event Handler duy nhất cho phần tử “document”, các phần tử bất kỳ bên dưới khi có sự kiện đều được chuyển tiếp lên cho phần tử “document” xử lý, trong quá trình xử lý thì sẽ biết được đối tượng tiêu điểm (**Target**) nào là nơi chính xác sự kiện xảy ra.

g. Client-side MVC Pattern (Mẫu MVC phía client)

Chúng ta đã tìm hiểu Server-side MVC Pattern được sử dụng rất phổ biến trong MPA ở phần trên. Tuy nhiên khi SPA ra đời thay thế MPA thì Server-side MVC Pattern không còn xài được nữa vì lúc này phía Server chỉ xử lý Web Request (Controller) rồi trả về dữ liệu (Model), còn phần View thì phía Web Browser lo. Cho nên MVC Pattern trong SPA đã dịch chuyển về phía Client-side, tức là Web Browser.

Nguyên lý hoạt động của Client-side MVC Pattern cũng tương tự Server-side MVC Pattern, chỉ khác 1 điểm là Model là dữ liệu nhận về từ REST API thay vì từ Business Layer.

h. Flux Architecture (Kiến trúc Flux)

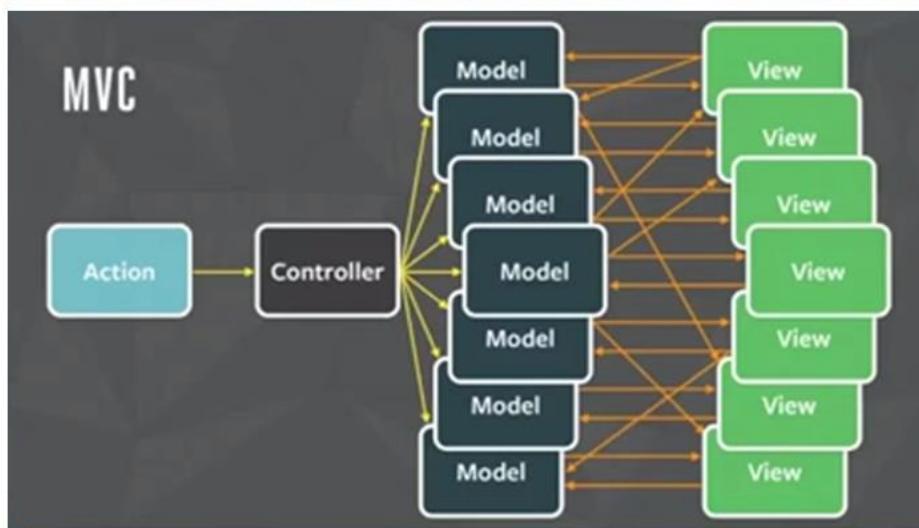
Bên cạnh những ưu điểm thì Client-side MVC Pattern có vài nhược điểm: khó dự đoán sự thay đổi trạng thái của ứng dụng do luồng dữ liệu hai chiều (**Bidirectional Data Flow**) cũng như debug.

Frontend MVC – Pros & Cons

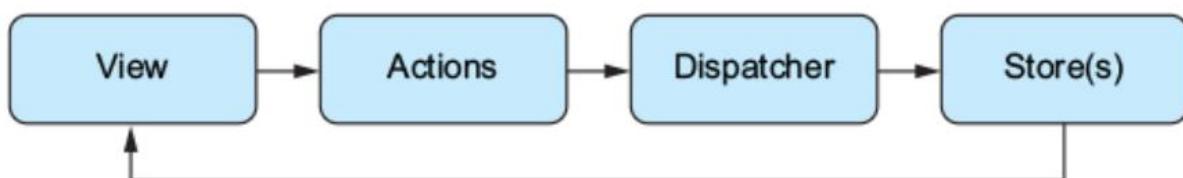
Pros:

1. separating presentation from model should be improving testability
2. separating view from controller most useful in web interfaces.

Cons: Hard to predict state changes due to bidirectional data flow and to debug



Do những nhược điểm này của Client-side MVC Pattern mà công ty Facebook đã đưa ra kiến trúc “Flux” dựa trên khái niệm luồng dữ liệu một chiều (**Unidirectional Data Flow**):



- **View:** là UI nhận dữ liệu từ Store.
- **Action:** là hành động được tạo ra khi User tương tác với hệ thống.
- **Dispatcher:** nhận Action rồi đẩy vào Store.
- **Store:** xử lý Action và thay đổi dữ liệu dựa trên loại Action -> các View được cập nhật theo tương ứng.

“Flux” chỉ là kiến trúc lý thuyết chứ không phải là Lib/FW. Các Lib/FW hiện thực “Flux” phổ biến có thể kể ra như “Alt”, “Fluxxor”...

i. Redux Library (Thư viện Redux)

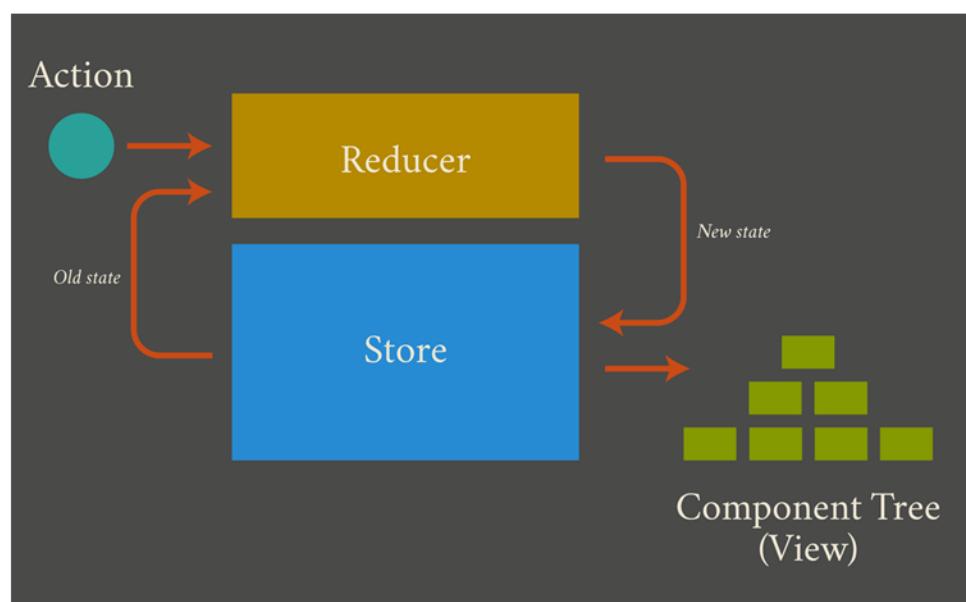
“Redux” là 1 hiện thực của kiến trúc “Flux” với 1 số chỉnh sửa khác với đặc tả trong “Flux”:

- “Redux” chỉ xài duy nhất 1 Store so với nhiều Stores trong “Flux”.
- “Redux” giới thiệu khái niệm “Middleware” để cung cấp thêm các tiện ích khác.
- “Redux” giới thiệu khái niệm “Reducer” để thay đổi dữ liệu của Store.

“Redux” được xem như là 1 predictable State Management Container, kết hợp với “React”, UI Lib phát triển bởi công ty Facebook, thành một cặp đôi hoàn hảo được sử dụng rất phổ biến trong cộng đồng FE Dev.

React – Redux Architecture

- Update App state: **The Store** dispatches an **Action** passed to Reducers.
- **Reducer must be a *pure function***: return a value, no AJAX, no DOM updates...



j. UI Transition (Chuyển đổi giao diện)

Khi đã nắm được kiến trúc FE, xây dựng UI, xử lý Event, công việc tiếp theo của FEP là chuyển đổi giao diện (**UI Transition**), tức là thay đổi UI hiện tại của User sang UI khác khi có một điều gì đó kích hoạt (**Trigger**).

Một vài ví dụ kích hoạt UI Transition:

- User đang ở LoginUI: khi User đăng nhập thành công, HomepageUI thay thế LoginUI.
- User đang ở ProductSearchUI: khi User bấm vào xem 1 Product, ProductViewUI sẽ thay thế ProductSearchUI.
- Tiếp tục User bấm nút “Edit” trong ProductViewUI -> ProductEditUI hiện ra.

k. UI Data binding/unbinding (Nạp vào/Lấy ra dữ liệu trên UI)

Thông thường một UI bao gồm 2 phần: các UI Element và dữ liệu. Khi User truy cập vào một UI thì các UI Element sẽ hiện lên trước, sau đó dữ liệu được lấy từ BE rồi nạp vào UI (**UI Data Binding**). Rồi khi User chỉnh sửa gì đó và lưu sự thay đổi thì hệ thống sẽ lấy dữ liệu trên UI (**UI Data Unbinding**) gửi qua BE lưu xuống DB.

Tiếp tục với ví dụ trên:

- User bấm nút “Edit” trong ProductViewUI -> ProductEditUI hiện ra các UI Element trước, dữ liệu về Product đang chọn được lấy từ BE về rồi nạp vào ProductEditUI.
- User chỉnh sửa thông tin của Product, bấm nút “Save” -> hệ thống lấy dữ liệu đã được User chỉnh sửa trên ProductEditUI ra và gửi qua BE để cập nhật cho Product đang sửa.

l. UI Paging Bar (Thanh phân trang UI)

Tương tự BEP hiện thực Data Search/Sorting/Paging, FEP ngoài việc cung cấp tính năng tìm kiếm dữ liệu theo từ khóa, sắp xếp dữ liệu còn hiện thực UI Paging Bar để giúp User phân trang dữ liệu nhằm tìm được dữ liệu nhanh chóng.



Một UI Paging Bar có các thành phần sau:

- **First**: nút nhấn đi về trang đầu tiên
- **Previous**: nút nhấn đi về 1 trang trước đó (so với trang hiện tại)
- **Next**: nút nhấn đi về 1 trang sau đó (so với trang hiện tại)
- **Last**: nút nhấn đi về trang sau cùng
- **Current Page**: trang hiện tại, có thể thay đổi để đi đến trang mong muốn
- **Page Size**: thay đổi kích thước trang, tức là số phần tử hiển thị trong 1 trang

m. Error Handling (Xử lý lỗi)

Tương tự như BEP, FEP cũng phải thực hiện Error Handling kỹ lưỡng.

n. UI Routing (Định tuyến UI)

Với MPA, mỗi trang Web được xác định bởi duy nhất 1 URL nên FEP không cần xử lý việc định tuyến UI (**UI Routing**). Tuy nhiên, với SPA chỉ có duy nhất 1 trang Web thì FEP phải thực hiện UI Routing để ánh xạ URL nào ứng với thành phần UI (**UI Component**) nào. Ví dụ: ~/orders -> OrderListComponent, ~/orders/:id -> OrderViewComponent

o. Browsing History (Lịch sử duyệt Web)

Web Browser hỗ trợ User các nút “Back” để xem lại 1 trang Web đã xem trước đó, nút “Forward” xem tới 1 trang đã xem rồi, và nút “Refresh” tải lại trang hiện tại. Những nút này hoạt động tốt với MPA nhưng không hoạt động được với SPA. Do đó, FEP phải làm cho các nút này hoạt động với SPA bằng cách sử dụng cơ chế UI Routing ở trên.

p. State Management (Quản lý trạng thái): UI State vs App State

Phía FE tồn tại 2 trạng thái mà chúng ta phải phân biệt rõ:

- **UI State** (Trạng thái giao diện người dùng): là tình trạng của các UI Element, ví dụ: Checkbox A đang được chọn, Checkbox B không được chọn, Radio Button C1 đang được chọn trong nhóm Radio Button Ci, phần tử thứ 6 trong Combo Box D1 đang được chọn...
- **App State** (Trạng thái ứng dụng): là tình trạng dữ liệu của ứng dụng đang được nạp vào các UI, ví dụ: danh sách các ngân hàng ứng dụng đang có là Bank 1, Bank 2, và Bank 3, danh sách này đang được sử dụng trong các UI 1, UI 3, UI 6, và UI 8, giả sử User dùng BankEditUI để thêm mới 1 ngân hàng/thay đổi thông tin 1 ngân hàng thì theo lô-gic các UI 1, UI 3, UI 6 và UI 8 phải cập nhật lại danh sách ngân hàng chúng đang xài.

Tùy theo yêu cầu của SW mà chúng ta sẽ quản lý UI State và/hoặc App State để đảm bảo tính toàn vẹn của hệ thống.

q. Asynchronous JavaScript and XML – AJAX Request

Web Browser được thiết kế chạy đơn luồng, nghĩa là nó không thể làm nhiều việc cùng một lúc, mỗi lần User thực hiện hành động nào đó thay đổi dữ liệu thì nó phải tải lại cả trang Web rất là mất thời gian, đôi khi dữ liệu thay đổi tí xíu.

Do đó công nghệ AJAX ra đời giúp việc truyền/nhận dữ liệu giữa Web Browser và Web Server được thực hiện một cách bất đồng bộ, không cần phải tải lại cả trang Web cũng như không ảnh hưởng các hành động khác của User.

Khi SPA ra đời thì công nghệ AJAX đóng vai trò chính yếu để thực hiện việc truyền/nhận dữ liệu giữa FE & BE.

Những ứng dụng hiện nay sử dụng kiểu dữ liệu là JSON trong AJAX Request thay vì XML.

r. Http Client (Thành phần truy cập dịch vụ Web)

Đây là 1 thành phần tiện ích mà FEP cần xây dựng để giúp cho việc lấy dữ liệu từ BE cũng như từ các API khác (ví dụ như lấy thông tin chứng khoán của 1 thị trường nào đó, lấy giá bán của 1 sản phẩm của 1 sàn nào đó) tiện dụng hơn.

s. User eXperience – UX (Trải nghiệm người dùng)

Ngoài việc xây dựng các UI đẹp, chạy đúng thì chúng ta cũng cần phải quan tâm đến tính tiện dụng của UI để đem đến trải nghiệm người dùng (**User eXperience – UX**) tốt nhất thì User mới thích mà quay lại xài SW, nếu không thì có thể SW có tính năng tốt nhưng họ xài thử với trải nghiệm tồi thì họ sẽ không quay lại.

Ví dụ đơn giản là thay vì đặt 1 cái Button ở trên đầu trang Web thì đặt ở dưới cùng của trang (hoặc thêm 1 cái Button có chức năng tương tự) sẽ tạo sự thuận tiện hơn cho User tương tác với hệ thống.

t. User Authentication (Định danh người dùng)

Để định danh người dùng trong FEP, thông thường mẫu đăng nhập (**Login Form**) được sử dụng cho phép User nhập vào thông tin đăng nhập (username/email/phone & password).

Sau khi User cung cấp thông tin và thực hiện đăng nhập thì FE sẽ gọi API kiểm tra thông tin của User:

- Nếu thông tin sai thì API sẽ gửi lỗi cho FE hiển thị cho User thấy.
- Nếu thông tin đúng thì User đăng nhập thành công, API sẽ trả về thông tin bảo mật (id hoặc token) để FE lưu trữ và sử dụng cho các yêu cầu của User sau này.

u. User Authorization (Phân quyền người dùng)

Sau khi User đăng nhập thành công, trong thông tin bảo mật API trả về có 1 danh sách các vai trò (**Role**) hoặc quyền (**Permission**) qui định những UI nào User được thấy và các chức năng nào User được phép sử dụng.

v. Internationalization – I18n (Đa ngôn ngữ)

Việc hiện thực đa ngôn ngữ phía FE chủ yếu cho các phần tử UI tĩnh như Label, Title (Button, Dialog, Window)...

w. Unit Test (Kiểm tra đơn vị)

Tính năng Unit Test trong FEP có ý nghĩa tương tự như BEP.

x. Self-Experience

Xây dựng phần FE sử dụng các REST API được cung cấp bởi phần BE của hệ thống eCommerce cơ bản (phần Self-Experience ở chương Backend Programming).

Về UI sẽ có 2 phần:

- Phần cho vai trò ADMIN
- Phần cho vai trò CUSTOMER

4) Non-Functional Requirement – NFR (Yêu cầu phi chức năng)

a. System Performance (Hiệu năng hệ thống)

Phát triển SW ngoài việc xử lý FR để cung cấp các tính năng mà User mong muốn một cách đúng đắn nhất thì cũng phải xử lý NFR để hệ thống IT hoạt động hiệu quả nhất, đáp ứng được nhu cầu của User đem đến User Experience tốt nhất.

Một trong các NFR khá quan trọng là hiệu năng hệ thống (**System Performance**). Tính chất này đảm bảo hệ thống IT chạy đúng khả năng của nó mà không bị chậm, ảnh hưởng đến User.

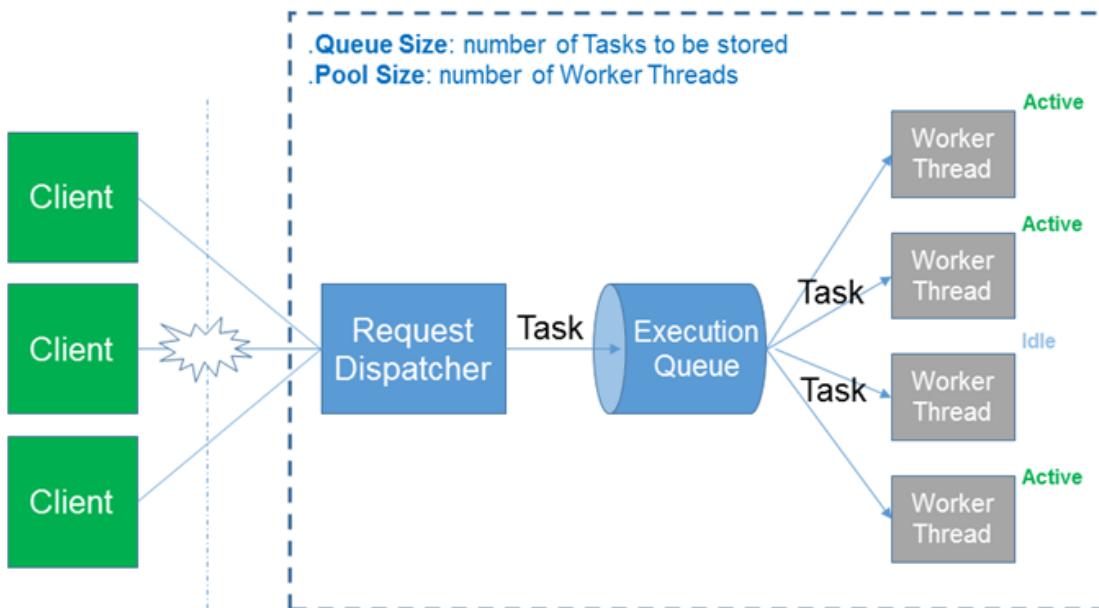
Sau đây tôi chia sẻ một số vấn đề về System Performance phổ biến.

1. **Bloated FE:** Phần FE quá cồng kềnh không cần thiết như:

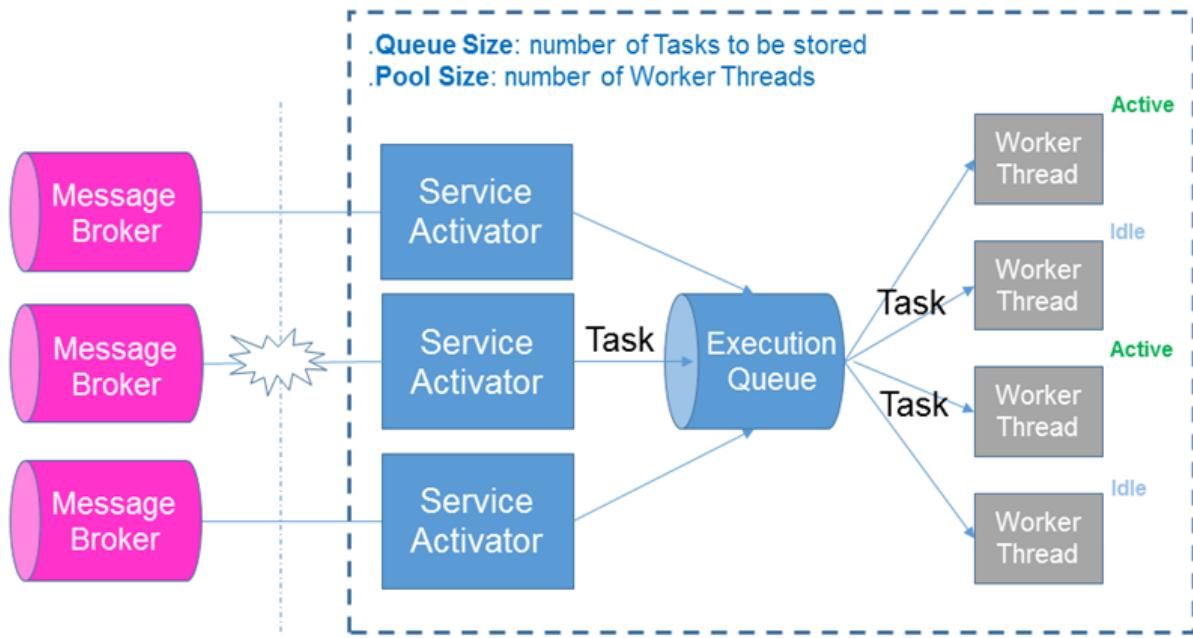
- Quá nhiều hình ảnh và/hoặc hình ảnh có kích thước quá lớn.
- Sử dụng không đúng bộ nhớ đệm của trình duyệt (**Browser Cache**).
- Lạm dụng quá nhiều Javascript Code hoặc AJAX Request.

2. **Thread Pool of Web Server:** Mỗi SW khi chạy trong OS có 1/nhiều tiến trình (**Process**), có thể có nhiều tiểu tiến trình (**Thread**) giúp xử lý các tác vụ song song, trong đó tồn tại 1 Thread chính gọi là Main Thread.

Trong trường hợp Web Server, khi có nhiều Web Request đến Web Server, nếu Web Server chỉ có 1 Main Thread thì không thể phục vụ nhiều User cùng lúc được thì sẽ làm Web Server chậm, nên nhiều Thread được sử dụng để xử lý vấn đề này. Tuy nhiên mỗi lần có Web Request đến mà tạo 1 Thread thì rất tốn tài nguyên và làm chậm hệ thống, việc tái sử dụng Thread là cách làm hợp lý hơn. Khái niệm **Thread Pool** xử lý vấn đề này.



3. **Thread Pool in System Integration:** Ngoài trường hợp Web Server thì Thread Pool cũng được xài trong các bài toán về System Integration sử dụng Messaging Queue.



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.

4. Memory Leak: Thất thoát bộ nhớ

In C/C++, a memory leak occurs when you allocate memory, assign the address of that memory to a reference variable, and then delete the reference to that memory without first de-allocating that memory.



In Java, memory leaks occur when a Java application inadvertently maintains a reference to an object that it does not ever intend to use again. There is no definitive way for the garbage collector to assess the intentions of the developer who built the application.



Vấn đề thất thoát bộ nhớ là vấn đề cực kỳ phổ biến khi lập trình, cho nên cần có các chiến lược xử lý một cách tốt đẹp nhất.

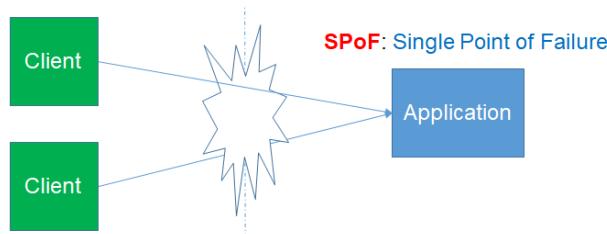
5. **Structured Query Language – SQL:** Vấn đề này liên quan đến việc viết các câu lệnh SQL không tốt, dẫn đến chạy rất chậm làm giảm System Performance.
6. **N+1 Select Problem:** Nếu bạn dùng giải pháp ORM cho phần tương tác với DB quan hệ thì lưu ý vấn đề “N+1 Select” khi bạn lấy danh sách dữ liệu và các tập con của các phần tử trong danh sách đó.

b. System Availability (Độ sẵn sàng hệ thống)

NFR thứ hai cũng khá là quan trọng là độ sẵn sàng của hệ thống (**System Availability**), nói đến khả năng hệ thống đáp ứng yêu cầu của User ở mức nào.

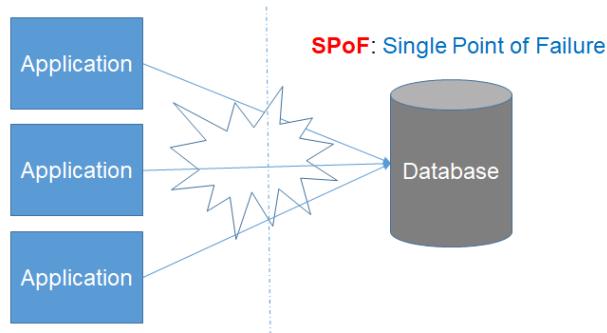
- Nếu System Availability = 100%: User truy cập được hệ thống ở mọi thời điểm, đây là điều kiện lý tưởng.
- Nếu System Availability < 100%: tồn tại một thời điểm nào đó mà hệ thống ở trạng thái nghỉ (**DOWN Status**), có nghĩa là User không sử dụng được hệ thống, sau đó thì hệ thống sẽ quay lại trạng thái hoạt động (**UP Status**) để phục vụ tiếp User.

Để hạn chế việc hệ thống rơi vào DOWN Status, chúng ta phải hạn chế các điểm đơn chẽt (**Single Point of Failure – SPoF**), đây là các điểm mà khi một trong chúng bị DOWN thì cả hệ thống sẽ bị DOWN luôn.



Ví dụ: 1 hệ thống gồm 1 Server A chạy FE, 1 Server B chạy BE và 1 Server C chạy DB -> Server A, Server B, và Server C được xem là các SPoF vì:

- Nếu Server A bị DOWN thì User không truy cập được UI -> hệ thống DOWN
- Nếu Server B bị DOWN thì UI không truy cập được REST API -> hệ thống DOWN
- Nếu Server C bị DOWN thì BE không lưu/đọc được dữ liệu từ DB -> hệ thống DOWN



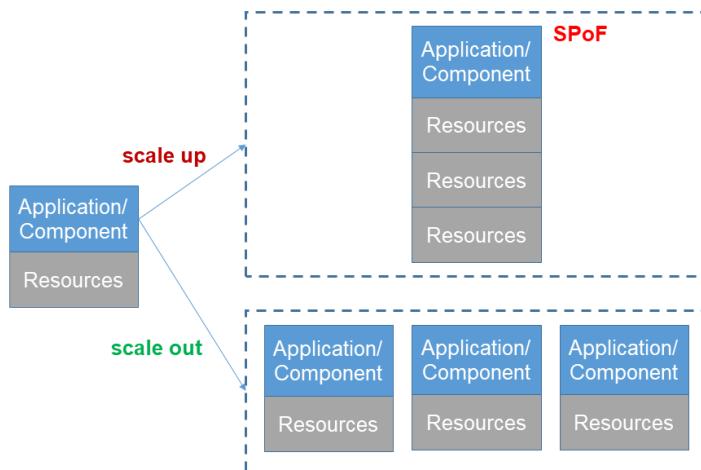
c. System Scalability (Việc co giãn hệ thống)

Hãy thử hình dung chúng ta xây dựng 1 SW cung cấp các FR tuyệt vời, chạy với System Performance rất tốt, và luôn sẵn sàng đáp ứng các yêu cầu của User, nhưng hệ thống chỉ phục vụ được cho 1000 User truy cập đồng thời (**Concurrent Request**), yêu cầu của các User từ thứ 1001 trở đi sẽ phải chờ hoặc có thể bị hủy.

Vậy thì làm sao để hệ thống có thể đáp ứng thêm Concurrent Request?

Đây là tiêu chí thứ ba của NFR mà chúng ta cần quan tâm: việc co giãn hệ thống (**System Scalability**). Có 2 phương pháp co giãn:

1. **Scale Up** (Co giãn theo chiều dọc): triển khai ứng dụng/thành phần trên 1 Server, tăng thêm tài nguyên cho Server để nó mạnh thêm.



2. **Scale Out** (Co giãn theo chiều ngang): triển khai ứng dụng/thành phần trên nhiều Server. Khi thực hiện cách này phải lưu ý đến trạng thái của yêu cầu (**Request State**), ví dụ: 1 Web Browser gửi 1 Web Request R1 đến Web Server để đăng nhập, giả sử R1 đến Server 1 đăng nhập hệ thống thành công, sau đó Web Browser gửi tiếp 1 Web Request R2 để lấy dữ liệu, R2 không đến Server 1 mà lại đến Server 2 -> Server 2 không biết là Web Browser kia đã đăng nhập hệ thống rồi nên sẽ báo lỗi R2 không hợp lệ.

d. System Security (Bảo mật hệ thống)

NFR phỏng biển cuối cùng là bảo mật hệ thống (**System Security**) bao gồm 2 phần là User Authentication & User Authorization đã được chia sẻ trong các phần BEP & FEP.

e. Self-Experience

Hiện thực 4 NFR ở trên cho hệ thống eCommerce cơ bản đã làm trong 2 phần Self-Experience trong 2 chương về BEP & FEP.

5) Reverse Engineering – REN (Kỹ thuật học ngược)

Khi chúng ta tham gia một dự án phần mềm đã chạy được một số Version đáng kể hoặc tiếp nhận một dự án phần mềm cũ để tiếp tục bảo trì và phát triển, việc đầu tiên chúng ta phải làm là tìm cách nắm được toàn bộ hệ thống phần mềm càng nhanh càng tốt.

Có 2 tình huống sẽ xảy ra:

- Dự án có đầy đủ tài liệu phân tích, thiết kế cũng như triển khai hệ thống: chúng ta rất là may mắn, việc của chúng ta chỉ là đọc hết tất cả tài liệu để hiểu hệ thống đang làm gì (WHAT) và làm như thế nào (HOW), nếu biết được cả tại sao (WHY) làm như vậy thì càng tốt.
- Dự án không có tài liệu phân tích, thiết kế cũng như triển khai hệ thống (hoặc tài liệu rất sơ sài không đủ thông tin cho người chưa từng làm dự án có thể đọc hiểu): chúng ta cần sử dụng kỹ thuật học ngược (**Reverse Engineering**) để hiểu hệ thống đang làm gì (WHAT) và làm như thế nào (HOW) cũng như tại sao (WHY) làm như thế. Tôi đã từng rơi vào tình huống này khi đảm nhận dự án Awesm từ 1 Tech Startup của Mỹ (tham khảo cuốn hồi ký “Hành trình ĐAM MÊ IT” của tôi).

a. REN Process (Qui trình học ngược)

Chúng ta sẽ sử dụng mô hình “4+1 Views” trong qui trình học ngược:

- Logical View:
 - Xác định các Business Flow bằng cách tương tác trên các UI và đọc Code
 - Xác định các Business Rule khi sử dụng UI, kiểm tra lại trong Code
 - Xác định các vai trò User tương tác với hệ thống
 - Trích xuất Data Model từ DB (ERD nếu là DB quan hệ)
 - Xác định các Entity trong Problem Space (nếu là ERD thì nhớ loại bỏ các bảng trung gian của quan hệ nhiều-nhiều)
 - Xác định các mối quan hệ bao đóng (**Composition Association**) giữa các Entity.
 - Xác định các mối quan hệ bao gộp (**Aggregation Association**) giữa các Entity.
 - Xác định các mối quan hệ thừa kế (**Inheritance Association**) giữa các Entity.
 - Với mỗi mối quan hệ, xác định lượng số (**Association Cardinality**).
 - Vẽ Domain Model và các State Diagram(nếu có).

- Development View:
 - Xác định những thành phần chính của hệ thống dựa trên các Business Flow.
 - Xác định các Logical Connection giữa những thành phần chính.
 - Vẽ Component Model và Architecture Model.
 - Xác định Technology Stack được sử dụng trong hệ thống.
- Process View:
 - Tách biệt các Business Flow.
 - Vẽ Flow Chart cho mỗi Business Flow.
- Physical View:
 - Truy cập vào Production Environment thu thập thông tin về các Servers
 - Vẽ Deployment Model
 - Xác định công nghệ triển khai
- User Stories:
 - Xác định các User Story dựa trên các Business Flow
 - Xác định danh sách Feature và gán các User Story vào mỗi Feature

b. Self-Experience

Áp dụng REN Process cho 1 dự án phần mềm bạn đã làm qua, giả sử dự án này không có bất cứ tài liệu nào cả.

6) CSI Formula (Công thức CSI)

Thế giới IT là cực kỳ rộng lớn, kiến thức công nghệ bao la như biển và thay đổi rất nhanh trong khi thời gian của mỗi chúng ta là hữu hạn. Vì vậy, có được một phương pháp phù hợp tiếp cận với các công nghệ sẽ giúp chúng ta ít bị lạc hậu.

Sau một quá trình khá dài trải nghiệm nhiều công nghệ trong ngành IT, tôi nhận thấy hầu hết công nghệ đều có một công thức chung mà tôi đặt tên là công thức CSI: xuất phát điểm là khái niệm (**Concept**) -> từ đó mới phát triển lên thành các đặc tả (**Specification – Spec**) -> rồi cuối cùng là các hiện thực (**Implementation – Impl**) khác nhau.

Như vậy, phần tử cốt lõi đầu tiên khi chúng ta tìm hiểu một công nghệ chính là **Concept**. Đây là chìa khóa để chúng ta rút ngắn thời gian khám phá các công nghệ.

a. Concept – Spec – Impl (Khái niệm – Đặc tả – Hiện thực)

Một Concept là kết quả của quá trình chuyển đổi từ các ý tưởng sang một mục tiêu cụ thể, ví dụ một số Concept trong Software Engineering sau:

- Client-side scripting
- AJAX
- Dynamic Web Page
- Web Service
- Business Transaction
- Object Relational Mapping (ORM)

Một Spec là một đặc tả chi tiết cho một Concept, ví dụ:

- ECMAScript là 1 Spec của Concept “Client-side Scripting”.
- JSP là 1 Spec của Concept “Dynamic Web Page”.
- JPA là 1 Spec của Concept “ORM”.

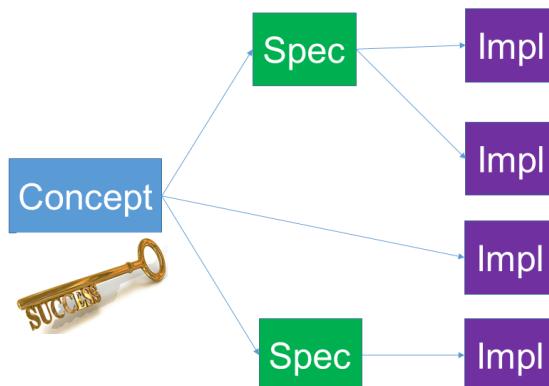
Mỗi Concept có thể có nhiều Spec, ví dụ:

- JSP là 1 Spec của Concept “Dynamic Web Page”.
- ASP.Net là 1 Spec của Concept “Dynamic Web Page”.
- PHP là 1 Spec của Concept “Dynamic Web Page”.

Một Spec có thể được hiện thực bởi nhiều công ty/nhóm/cá nhân tạo ra nhiều Implementation, ví dụ:

- Javascript of Mozilla là một Implementation của Spec “ECMAScript”.
- Jscript of Microsoft là một Implementation của Spec “ECMAScript”.
- Glassfish of Oracle là một Implementation của Spec “JSP” và “JPA”.
- WebSphere of IBM là một Implementation của Spec “JSP” và “JPA”.
- Hibernate JPA là một Implementation của Spec “JPA”.

Mô hình diễn tả mối quan hệ giữa Concept – Spec – Impl được biểu diễn qua hình sau:



Ví dụ:

- Object-Relational Mapping (ORM) là 1 Concept
- Java Persistence API (JPA) là 1 Spec cho Concept ORM
- OpenJPA, Hibernate JPA, EclipseLink là những Implementation cho Spec JPA

Từ ví dụ trên chúng ta thấy là thay vì nhảy vô học các Implementation cụ thể như OpenJPA, Hibernate JPA, EclipseLink ... sẽ mất rất nhiều thời gian, chúng ta nên tìm hiểu Concept ORM trước, sau đó xem qua Spec JPA, khi đã nắm được Concept & Spec thì chỉ cần học sử dụng 1 Implementation như Hibernate JPA, sau này có việc phải chuyển qua xài những Implementation khác như OpenJPA, EclipseLink sẽ tốn ít thời gian vì chúng tương tự nhau, nếu có khác thì chỉ khác nhau chút đỉnh.

Lưu ý: không nhất thiết mọi Concept đều phải có Spec (hình trên có mũi tên đi thẳng từ Concept đến Implementation), đôi khi một Spec của một Concept không có tên hoặc không được biết đến, khi đó Implementation được sử dụng thay cho Spec, ví dụ:

- Hibernate (native) là một Implementation của Concept “ORM”.
- Entity Framework là một Implementation của Concept “ORM”.

b. Self-Experience

Cho một danh sách các thuật ngữ sau:

- Eclipse
 - MS Visual Studio
 - Hibernate
 - EclipseLink
 - JPA
 - JSP
 - Spring
 - MSSQL
 - MySQL
 - Oracle DB
 - IoC
 - JDBC
 - Spring Transaction
 - JDBC Template
 - MongoDB
 - Redis
 - ASP.Net
 - Castle
 - Entity Framework
 - Memcached
 - Riak
1. Hãy sắp xếp chúng vào 1 bảng có 3 cột: Concept-Spec-Impl.
 2. Sau đó điền vào những chỗ còn trống cho đầy bảng, nghĩa là mỗi dòng sẽ có đủ 3 thành phần Concept-Spec-Impl.

(Đáp án nằm ở trang kế)

Đáp án câu 1:

Concept	Specification	Implementation
		Eclipse, Netbeans
	JPA	Hibernate, EclipseLink
IoC		Spring, Castle
		MSSQL, MySQL, Oracle DB
		MongoDB, Memcached, Riak, Redis
	JDBC	
	Spring Transaction	
	JDBC Template	
	ASP.Net	
		Entity Framework

Đáp án câu 2:

Concept	Specification	Implementation
IDE		Eclipse, MS Visual Studio
ORM	JPA	Hibernate, EclipseLink
	Entity Framework	MS .Net Framework
IoC	Dependency Injection	Spring, Castle
DBMS	RDBMS	MSSQL, MySQL, Oracle DB
	NoSQL	MongoDB, Memcached, Riak, Redis
Database Connectivity	JDBC	Java SE
Business Transaction	Spring Transaction	Spring Framework
Data Access Template	JDBC Template	Spring Framework
Dynamic Web Page	JSP	Tomcat, Glassfish
	ASP.Net	IIS

Phụ Lục

I) Abbreviation (Các từ viết tắt trong bài)

AI	Artificial Intelligence
AJAX	Asynchronous JavaScript and XML
ALU	Athrimetical Logical Unit
AOP	Aspect-Oriented Programming
API	Application Programming Interface
BA	Business Analyst
BE	Backend
BEP	Backend Programming
CNE	Computer Network
CPT	Computer
CPU	Central Processing Unit
CRUD	Create/Read/Update/Delete
CSI	Concept-Specification-Implementation
CSS	Cascading Style Sheet
CSSW	Closed-Source Software
CST	Computer Structure
DB	Database
DBA	Database Administrator
DBMS	DataBase Management System
DCL	Data Control Language
DDL	Data Definition Language
DI	Dependency Injection
DML	Data Manipulation Language
DSC	Data Science
DST	Data Structure
ERD	Entity Relationship Diagram
ERP	Enterprise Resource Planning
FE	Frontend
FEP	Frontend Programming
FP	Functional Programming
FR	Functional Requirement
FW	Framework

HD	Hard Disk
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
IoC	Inversion of Control
ISP	Internet Service Provider
LAN	Local Area Network
Lib	Library
MAN	Metropolitain Area Network
MPA	Multiple Page Application
MVC	Model-View-Controller
NFR	Non-Functional Requirement
OOP	Object-Oriented Programming
ORM	Object Relational Mapping
OS	Operating System
OSSW	Open-Source Software
PAN	Personal Area Network
PL	Programming Language
PM	Project Manager
PMO	Programming Model
PP	Procedure Programming
QA	Quality Assurance
QC	Quality Control
RAM	Random Access Memory
REN	Reverse Engineering
REST	REpresentational State Transfer
ROM	Read Only Memory
RPC	Remote Procedure Call
SA	Software Architect
SaaS	Software as a Service
SWAR	Software Architecture
SD	Software Developer
SDLC	Software Development Life Cycle
SME	Small and Medium Enterprise
SP	Software Professional
SPA	Single Page Application
SPoF	Single Point of Failure

SQL	Structured Query Language
SW	Software
SWEN	Software Engineering
TCL	Transaction Control Language
TPG	Translating Program
UI	User Interface
URL	Uniform Resource Locator
US	User Story
UT	Unit Test
UX	User Experience
VCS	Version Control System
WAN	Wide Area Network
XML	Extensible Markup Language

2) Popular English Terms (Các thuật ngữ English thông dụng)

Abstraction	Trùu tượng hóa
Access Control	Điều khiển truy cập
Activity Table	Bảng ghi hoạt động
Adaptability	Tính thích ứng
Aggregate Function	Hàm tổng hợp
Aggregation Association	Quan hệ bao gộp
Agile Methodology	Phương pháp nhanh & lặp
Allocation Strategy	Chiến lược cấp phát
Anonymous Class	Lớp nặc danh
App State	Trạng thái ứng dụng
Application Layer	Lớp ứng dụng
Application Software	Phần mềm ứng dụng
Architecture Model	Mô hình kiến trúc
Architecture Review Board – ARB	Ban duyệt kiến trúc
Artificial Intelligence	Trí tuệ nhân tạo
Aspect-Oriented Programming – AOP	Lập trình khía cạnh
Assembly	Hợp ngữ
Association Cardinality	Lượng số quan hệ
Asynchronous	Bất đồng bộ
Asynchronous Integration	Tích hợp bất đồng bộ
Asynchronous Processing	Xử lý bất đồng bộ
Athrimetical Logical Unit – ALU	Đơn vị xử lý luận lý số học
Atomicity	Đơn nhất
Attitude	Tư duy
Automation Test	Kiểm tra tự động
Back Door	Cửa hậu
Backend Programming – BEP	Lập trình phía Backend
Background	Nền tảng
Batch Processing	Xử lý theo lô
Behavior	Hành vi
Bidirectional Data Flow	Luồng dữ liệu hai chiều
Binary System	Hệ nhị phân
Black Box	Hộp đen
Block	Khối
Browser Cache	Bộ nhớ đệm của trình duyệt

Browsing History	Lịch sử duyệt Web
Bug Fixing	Việc sửa lỗi
Bug or Defect	Lỗi
Business Analyst – BA	Nhà phân tích nghiệp vụ
Business Flow	Luồng nghiệp vụ
Business Layer	Lớp nghiệp vụ
Business Rule	Qui luật nghiệp vụ
Business Transaction	Giao dịch nghiệp vụ
Call Stack	Chồng các cuộc gọi
Can-do	(Tư duy) Có thể làm
Central Processing Unit – CPU	Bộ xử lý trung tâm
Class	Lớp
Class Method	Phương thức lớp
Class Variable	Biến lớp
Client	Máy khách
Client-Server Model	Mô hình Khách-Chủ
Closed-source Software	Phần mềm nguồn đóng
Cloud Solution	Giải pháp đám mây
Code Conflict	Xung đột mã
Code Quality	Chất lượng mã
Code Reuse	Tái sử dụng Code
Code Review	Xem lại code
Coding Convention	Qui định viết code
Coding Standard	Tiêu chuẩn về viết code
Coding Style	Định dạng code
Command Line Interface – CLI	Giao diện dòng lệnh
Communication	Giao tiếp
Communication Protocol	Giao thức giao tiếp
Compiler	Trình biên dịch
Component Model	Mô hình thành phần
Composition Association	Quan hệ bao đóng
Compound Noun	Danh từ ghép
Computer	Máy vi tính
Computer Device	Thiết bị máy tính
Computer History	Lịch sử máy tính
Computer Network	Mạng máy tính

Computer Program	Chương trình máy tính
Computer Structure	Cấu trúc máy tính
Computer Type	Loại máy tính
Computer-User Interaction	Tương tác người dùng với máy tính
Concept	Khái niệm
Conclusion	Kết luận
Concurrency Handling	Xử lý đồng thời
Concurrent Access	Truy cập đồng thời
Concurrent Request	Yêu cầu đồng thời
Consistency	Nhất quán
Constant	Hằng
Continuous Delivery – CD	Liên tục triển khai phiên bản (version)
Continuous Integration – CI	Liên tục tích hợp tính năng
Control Flow	Luồng điều khiển
Control Unit – CU	Đơn vị điều khiển
Controller Layer	Lớp điều khiển
Cryptography	Mật mã hoá
Data	Dữ liệu
Data Analytics	Phân tích dữ liệu
Data Caching	Lưu dữ liệu tạm thời
Data Control Language – DCL	Ngôn ngữ điều khiển truy cập dữ liệu
Data Definition Language – DDL	Ngôn ngữ định nghĩa dữ liệu
Data Integrity	Tính toàn vẹn dữ liệu
Data Manipulation Language – DML	Ngôn ngữ xử lý dữ liệu
Data Mining	Khai thác dữ liệu
Data Paging	Phân trang dữ liệu
Data Science	Khoa học dữ liệu
Data Scientist	Nhà khoa học dữ liệu
Data Scope	Tầm vực dữ liệu
Data Search	Tìm kiếm dữ liệu
Data Sorting	Sắp xếp dữ liệu
Data Structure – DST	Cấu trúc dữ liệu
Data Type	Kiểu dữ liệu
Data Validation	Xác nhận dữ liệu
Data Warehouse	Kho dữ liệu
Data-Level Authorization	Phân quyền mức dữ liệu

Database	Cơ sở dữ liệu
Database Admin – DBA	Nhà quản trị CSDL
Database Commit	Thực thi trong cơ sở dữ liệu
Database Denormalization	Phá chuẩn hóa cơ sở dữ liệu
DataBase Management System – DBMS	Hệ quản trị CSDL
Database Norm	Dạng chuẩn cơ sở dữ liệu
Database Normalization	Chuẩn hóa cơ sở dữ liệu
Database Rollback	Hủy trong cơ sở dữ liệu
Database Transaction	Giao dịch cơ sở dữ liệu
Datalink Layer	Lớp kết nối dữ liệu
Dependency Injection – DI	Tiêm sự phụ thuộc
Deployment Model	Mô hình triển khai
Design Diagram	Lược đồ thiết kế
Design Pattern	Mẫu thiết kế
Desktop Computer	Máy tính để bàn
Development Team	Nhóm phát triển
Development View	Góc nhìn phát triển
Device Driver	Chương trình điều khiển thiết bị
DevOps	Chuyên viên triển khai vận hành hệ thống
Directory	Thư mục
Disk Management	Quản lý đĩa cứng
Domain Knowledge	Kiến thức lĩnh vực
Domain Model	Mô hình lĩnh vực
Domain Name	Tên miền
Domain Name Server – DNS	Máy định tên miền
Domain Name Service	Dịch vụ định tên miền
Duration	Lâu dài
Dynamic Content	Nội dung động
Dynamic Element	Phần tử động
Eager-to-learn Spirit	Tinh thần ham học hỏi
eCommerce	Thương mại điện tử
Employee	Nhân viên
Encapsulation	Tính bao đóng
Energy	Nguồn năng lượng
Enterpreneurial Spirit	Tinh thần làm chủ
Entity	Thực thể

Entity Relationship Diagram – ERD	Mô hình quan hệ thực thể
Entry point	Điểm vào
Environment	Môi trường
Error Handling	Xử lý lỗi
Event Bubbling	Chuyển tiếp sự kiện
Event Handler	Bộ xử lý sự kiện
Event Handling	Xử lý sự kiện
Event-Driven Programming	Lập trình dựa vào sự kiện
Exception Handling	Xử lý lỗi
Execution Flow	Luồng thực thi
Exit Point	Điểm kết thúc
Experience	Kinh nghiệm
Expression	Biểu thức
Extended Partition	Phân khu mở rộng
Fact	Dữ kiện
Features	Các tính năng
Fetch Strategy	Chiến lược nạp
File	Tập tin
Filter Chain	Chuỗi lọc
Flexibility	Tính linh hoạt
Flow Chart	Lược đồ dòng chảy
Formal Parameter	Thông số hình thức
Framework – FW	Khung làm việc
Frontend Programming – FEP	Lập trình phía Frontend
Function of Word	Chức năng của từ
Functional Programming	Lập trình hàm
Functional Requirement – FR	Yêu cầu chức năng
Fundamental Knowledge	Kiến thức nền tảng
Gateway	Máy trung gian
Generics	Tính tổng quát hóa
Goal	Mục tiêu
Graphic UI	Giao diện đồ họa
Hard Drive – HD	Đĩa cứng
Hardware	Phần cứng
High Adaptability	Tính thích ứng cao
Hub	Bộ khuếch đại

Hyper Text Markup Language – HTML	Ngôn ngữ liên kết văn bản
I/O Management	Quản lý nhập xuất
Implementation	Hiện thực
In-house Solution	Giải pháp trong nhà
Inheritance	Tính thừa kế
Inheritance Association	Quan hệ thừa kế
Input Device	Thiết bị nhập
Instance Method	Phương thức thực thể
Instance Variable	Biến thực thể
Integrated Development Environment – IDE	Môi trường phát triển tích hợp
Interface	Giao diện
Interior Node	Nút trung gian
Internet	Mạng toàn cầu
Internet Service Provider – ISP	Nhà cung cấp dịch vụ internet
Interpreter	Trình thông dịch
Inversion of Control – IoC	Đảo ngược điều khiển
IP Address	Địa chỉ IP
Isolation	Tách biệt
IT Outsourcing Companies – ITOC	Công ty gia công IT
IT System	Hệ thống IT
Job	Nghề
Kernel	Phần lõi
Key	Khóa
Keyboard	Bàn phím
Keyword	Từ khóa
Knowledge	Kiến thức
Language Syntax	Cú pháp ngôn ngữ
Laptop Computer	Máy tính xách tay
Leaf Node	Nút lá
Lexical Analysis	Phân tích từ vựng
Library – Lib	Thư viện
List	Danh sách
Local Area Network – LAN	Mạng nội bộ
Local Method	Phương thức cục bộ
Local Variable	Biến cục bộ
Logic	Tính hợp lý

Logical Connection	Kết nối luận lý
Logical Partition	Phân khu luận lý
Logical Separation	Tách biệt luận lý
Logical Structure	Cấu trúc luận lí
Logical Thinking	Suy nghĩ luận lý
Logical View	Góc nhìn luận lý
Login Form	Mẫu đăng nhập
Long-term Goal	Mục tiêu dài hạn
Machine Language	Ngôn ngữ máy
Main Frame	Máy tính lớn
Manual Test	Kiểm tra bằng tay
Map	Ánh xạ
Memory Block	Khối bộ nhớ
Memory Leak	Thất thoát bộ nhớ
Memory Management	Quản lý bộ nhớ
Message	Thông điệp
Messaging Queue	Hàng đợi chứa thông điệp
Method	Phương thức
Method Call	Cuộc gọi phương thức
Method Overloading	Phương pháp quá tải phương thức
Method Overriding	Phương pháp đè phương thức
Method Scope	Tầm vực của phương thức
Method-Level Authorization	Phân quyền mức phương thức
Metropolitant Area Network – MAN	Mạng nội thành
Micro Co-Processor – MCP	Bộ đồng xử lý
Mini Frame	Máy tính vừa
Mobile Device	Thiết bị di động
Mobile Programming	Lập trình ứng dụng di động
Monitor	Màn hình
Mouse	Chuột
Multi Programming	Xử lý đa chương trình
Multi Tasking	Xử lý đa nhiệm
Multi-Page Application – MPA	Ứng dụng đa trang
Native Speaker	Người bản xứ
Nature of Word	Bản chất của từ
Network Interface Card – NIC	Cạc giao tiếp mạng

Network Layer	Lớp mạng
Network Operating System	Hệ điều hành mạng
Network Security	Bảo mật trên mạng
Network Software	Phần mềm mạng
Non-Functional Requirement – NFR	Yêu cầu phi chức năng
NoSQL Database	CSDL NoSQL
Object	Đối tượng
Object Construction	Xây dựng đối tượng
Object Destruction	Hủy đối tượng
Object Graph	Đồ thị đối tượng
Object Instantiation	Tạo mới đối tượng
Object Life Cycle	Vòng đời của đối tượng
Object Relational Mapping – ORM	Ánh xạ quan hệ & đối tượng
Object State	Trạng thái đối tượng
Object Type	Kiểu đối tượng
Object-Oriented Programming – OOP	Lập trình hướng đối tượng
Open-Source Community	Cộng đồng mã nguồn mở
Open-Source Rule	Nguyên tắc nguồn mở
Open-Source Software	Phần mềm nguồn mở
Operand	Toán hạng
Operating System	Hệ điều hành
Operator	Phép toán/ Toán tử
Optimistic Locking	Khóa lạc quan
Output Device	Thiết bị xuất
Over Time – OT	Làm thêm ngoài giờ
Palmtop	Máy tính cầm tay
Parallel Development	Phát triển song song
Parallel Processing	Xử lý song song
Parameter	Thông số
Partition	Phân khu
PASSION	ĐAM MÊ
Performance	Hiệu năng
Persistence Layer	Lớp lưu trữ
Personal Area Network – PAN (Bluetooth)	Mạng cá nhân
Personal Computer – PC	Máy tính cá nhân
Pessimistic Locking	Khóa bi quan

Physical Connection	Kết nối vật lý
Physical Layer	Lớp vật lý
Physical Separation	Tách biệt vật lý
Physical Structure	Cấu trúc vật lí
Physical View	Góc nhìn vật lý
Placement Strategy	Chiến lược đặt
Pointer Type	Kiểu dữ liệu con trỏ
Polymorphism	Tính đa hình
Presentation	Thuyết trình
Presentation Layer	Lớp trình bày
Primary Partition	Phân khu chính
Primitive Type	Kiểu dữ liệu cơ bản
Printer	Máy in
Problem	Vấn đề
Problem Solving	Giải quyết vấn đề
Problem Space	Không gian vấn đề
Procedure	Thủ tục
Procedure Call	Cuộc gọi thủ tục
Procedure Programming	Lập trình thủ tục
Process	Tiến trình
Process Control Block – PCB	Bộ điều khiển tiến trình
Process Management	Quản lý tiến trình
Process View	Góc nhìn qui trình
Processing Device	Thiết bị xử lí
Processor Management	Quản lý CPU
Product Owner – PO	Chủ sản phẩm
Production Environment	Môi trường sử dụng
Productivity	Có năng suất
Professional English	Tiếng Anh làm việc
Professional Knowledge	Kiến thức chuyên ngành
Program Counter	Bộ đếm chương trình
Programmer	Lập trình viên
Programming	Lập trình
Programming Concept	Khái niệm trong lập trình
Programming Language	Ngôn ngữ lập trình
Programming Model – PMO	Mô hình lập trình

Programming Principle	Nguyên tắc lập trình
Project Manager – PM	Nhà quản lý dự án
Proof of Concept – POC	Thử nghiệm ý tưởng
Property	Thuộc tính
Pros/Cons	Ưu/nhược điểm
Protocol	Giao thức
Quality Assurance – QA	Kiểm tra chất lượng qui trình
Quality Control – QC	Kiểm tra chất lượng sản phẩm
Random Access Memory – RAM	Bộ nhớ truy cập ngẫu nhiên
Read Only Memory – ROM	Bộ nhớ chỉ đọc
Real Parameter	Thông số thực
Real-life Vocabulary	Từ vựng thực tế
Recursive Call	Cuộc gọi đệ quy
Relational Database	Cơ sở dữ liệu quan hệ
Release	Triển khai phần mềm
Remote Procedure Call – RPC	Gọi thủ tục từ xa
Request/Response Protocol	Giao thức hỏi đáp
Requirement Analysis	Phân tích phần mềm
Requirement Gathering	Thu thập yêu cầu phần mềm
Resource	Tài nguyên
Reusability	Tính tái sử dụng
Reverse Engineering – REN	Kỹ thuật học ngược
Role	Vai trò
Root Cause	Nguyên nhân chính
Root Node	Nút gốc
Router	Bộ tìm đường
Scale Out	Co giãn theo chiều ngang
Scale Up	Co giãn theo chiều dọc
Scenario	Tình huống
Scheduler	Bộ định thời
Security	Tính bảo mật
Self-Experience	Tự trải nghiệm
Semantic Processing	Xử lý ngữ nghĩa
Server	Máy chủ
Session Layer	Lớp phiên làm việc
Set	Tập hợp

Short-term Goal	Mục tiêu ngắn hạn
Single Page Application – SPA	Ứng dụng đơn trang
Single Point of Failure – SPoF	Điểm đơn chét
Skill	Kỹ năng
Slang Word	Từ lóng
Small and Medium Enterprises – SME	Công ty vừa và nhỏ
Smart Phone	Điện thoại thông minh
Soft Skill	Kỹ năng mềm
Software	Phần mềm
Software Architect – SA	Kiến trúc sư phần mềm
Software Architecture – SWAR	Kiến trúc phần mềm
Software as a Service – SaaS	Phần mềm là dịch vụ
Software Deployment	Triển khai phần mềm
Software Design	Thiết kế phần mềm
Software Developer – SD	Nhà phát triển phần mềm
Software Development Life Cycle – SDLC	Vòng đời phát triển phần mềm
Software Development Process	Qui trình phát triển phần mềm
Software Engineering	Công nghệ phần mềm
Software Implementation	Hiện thực phần mềm
Software Maintenance	Bảo trì phần mềm
Software Testing	Kiểm tra phần mềm
Solution	Giải pháp
Solution Space	Không gian giải pháp
Source Code	Mã nguồn
Speaker	Loa
Specification	Đặc tả
Stage Environment	Môi trường thử nghiệm
State	Trạng thái
State Digram	Lược đồ trạng thái
State Management	Quản lý trạng thái
Statement	Phát biểu
Static Content	Nội dung tĩnh
Static Element	Phần tử tĩnh
Storage Device	Thiết bị lưu trữ
Structured Data	Dữ liệu cấu trúc
Structured Query Language – SQL	Ngôn ngữ truy vấn có cấu trúc

Structured Type	Kiểu dữ liệu cấu trúc
Study	Học
Super Computer	Siêu máy tính
Supplemental Knowledge	Kiến thức bổ sung
Switch	Bộ chuyển mạch
Synchronous	Đồng bộ
Synchronous Integration	Tích hợp đồng bộ
Syntactical Analysis	Phân tích cú pháp
Syntax	Cú pháp
System Availability	Độ sẵn sàng của hệ thống
System Clock	Mạch xung nhịp hệ thống
System Improvement	Cải tiến hệ thống
System Integration	Tích hợp hệ thống
System Logging	Lưu vết hệ thống
System Performance	Hiệu năng hệ thống
System Scalability	Co giãn hệ thống
System Security	Bảo mật hệ thống
System Skeleton	Khung hệ thống
System Software	Phần mềm hệ thống
Target	Đối tượng
Team Lead – TL	Trưởng nhóm
Teamwork	Làm việc nhóm
Technology Stack	Chồng công nghệ
Test Case	Trường hợp kiểm tra
Thread	Tiêu tiến trình
Thread Pool	Hò chứa các tiêu tiến trình
Time Management	Quản lý thời gian
Transaction	Giao dịch
Transaction Control Language – TCL	Ngôn ngữ điều khiển giao dịch
Translating Program – TPG	Chương trình dịch
Transport Layer	Lớp vận chuyển
Tree	Cây
Trend	Xu hướng
Troubleshooting	Việc điều nghiên
Tutorial	Giáo trình tự học
Type	Kiểu

UI Component	Thành phần UI
UI Data binding	Nạp vào dữ liệu trên UI
UI Data unbinding	Lấy ra dữ liệu trên UI
UI Paging Bar	Thanh phân trang UI
UI Routing	Định tuyến UI
UI State	Trạng thái giao diện người dùng
Unidirectional Data Flow	Luồng dữ liệu một chiều
Unit of Measurement	Đơn vị đo lường
Unit Test – UT	Kiểm tra đơn vị
Unstructured Data	Dữ liệu không cấu trúc
User	Người dùng
User Authentication	Định danh người dùng
User Authorization	Phân quyền người dùng
User Experience – UX	Trải nghiệm người dùng
User Interface – UI	Giao diện người dùng
User Story	Tương tác người dùng
Value	Giá trị
Variable	Biến
Variable Scope	Tầm vực của biến
Version Control System – VCS	Hệ quản lý phiên bản
Virtual Machine	Máy ảo
Virtual Memory	Bộ nhớ ảo
Waterfall Methodology	Phương pháp thác nước
Wide Area Network – WAN	Mạng diện rộng
Wireless Technology	Công nghệ không dây

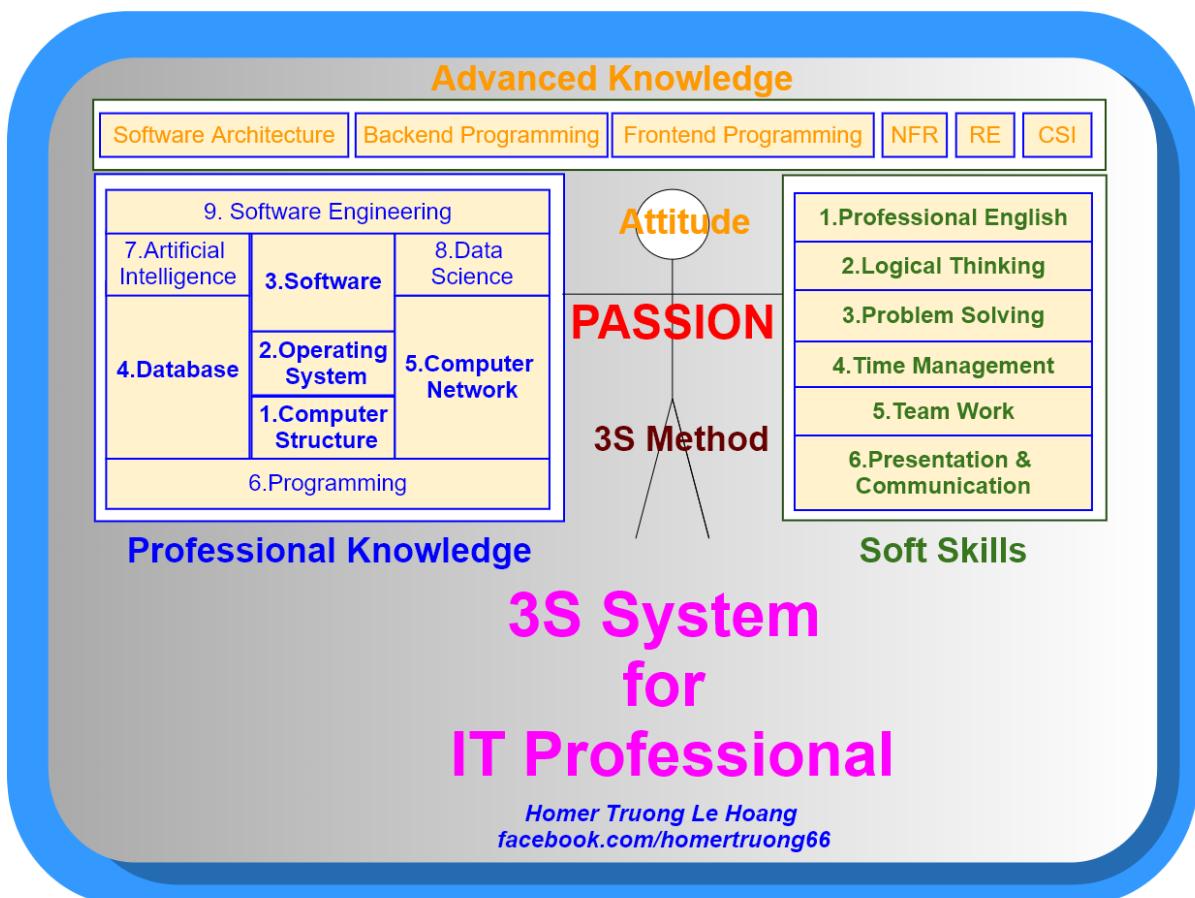
3) Software Development Toolkit (Bộ công cụ phát triển phần mềm)

- Flow Chart Tool: **Gliffy** (<https://chrome.google.com/webstore/detail/gliffy-diagrams/bhmicilclplefnflapjmnnngmkkkpfad?hl=en>)
- Git Client: **Git SCM** (<https://git-scm.com/downloads>)
- Git UI: **SourceTree** (<https://sourcetreeapp.com>), **SmartGit** (<https://syntevo.com/smartygit>)
- Java SE: **JDK** (<https://www.oracle.com/technetwork/java/javase/downloads/index.html>)
- Backend IDE: **IntelliJ** (<https://www.jetbrains.com/idea>)
- Java Build Tool: **Maven** (<https://maven.apache.org>)
- Java Web Server: **Jetty** (<https://eclipse.org/jetty/>), **Tomcat** (<http://tomcat.apache.org>)
- Database: **MySQL** (<https://dev.mysql.com/downloads>)
- Database Client: **HeidiSQL** (<https://www.heidisql.com>)
- API Testing: **Postman** (<https://www.getpostman.com>)
- Frontend IDE: **Visual Studio Code** (<https://code.visualstudio.com>)
- Static Web Server: **Apache HTTP Server** (<https://httpd.apache.org>)
- Reload Capability Server: **Live Server** (<https://www.npmjs.com/package/live-server>)
- Javascript Server: **Nodejs** (<https://nodejs.org>)
- Javascript Build Tool: **Yarn** (<https://yarnpkg.com>)
- Javascript Debug Tool: **React Developer Tools**
(<https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadoplbjfkapdkoienihi?hl=en>), **Redux DevTools**
(<https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmffibljd?hl=en>)
- Secure Shell Client: **Bitvise** (<https://www.bitvise.com>), **Putty** (<https://www.putty.org>)
- Data Caching: **Redis** (<https://redis.io>)
- Messaging Queue: **RabbitMQ** (<https://rabbitmq.com>), **Kafka** (<https://kafka.apache.org>)
- Virtual Machine: **Virtual Box** (<https://virtualbox.org>), **VMWare Workstation Player**
(<https://www.vmware.com/asean/products/workstation-player.html>)
- DevOps Build and Deployment Tool: **Jenkins** (<https://jenkins.io>)
- Load Balancing: **HAProxy** (<http://www.haproxy.org>)
- File Storage: **AWS S3 Browser** (<https://s3browser.com>)

Lời tổng kết

Trên đây tôi đã chia sẻ với bạn toàn bộ kiến thức/kinh nghiệm tổng hợp trong suốt một quá trình dài trải nghiệm ngành IT của tôi.

Đó là Yếu tố con người, Kiến thức chuyên môn, Kỹ năng mềm, và Kiến thức nâng cao được thể hiện một cách trực quan bằng Hệ thống 3S mà mỗi dân IT cần nắm vững để làm việc hiệu quả:



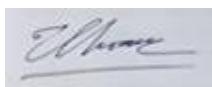
Hi vọng cuốn cẩm nang “Hệ thống 3S cho dân IT” này sẽ giúp ích cho bạn tiến nhanh và chắc trên con đường ĐAM MÊ IT.

Mọi ý kiến đóng góp của bạn cho cuốn cẩm nang ngày càng tốt hơn vui lòng gửi về địa chỉ email sau: homertruong66@gmail.com.

Xin cảm ơn!

Nếu 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 bạn đạt được nhiều thành quả trong ngành IT và hạnh phúc trong cuộc sống!



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/trunglehoang>