



TOIDICODEDAO BLOG

facebook.com/nguyenpht tổng hợp

MỤC LỤC

MỤC LỤC	1
About me.....	18
Về bản thân	18
Về blog này	20
Hỏi đáp.....	21
Short-circuit evaluation : Đoản mạch và chập điện	23
Học ngôn ngữ lập trình nào bây giờ (3 Phần)	26
Phần 1 – Trước khi hỏi câu này, hãy tự hỏi : Minh muốn học lập trình để làm gì?	26
Phần 2 – Lựa chọn thật ra không quan trọng, học một ngôn ngữ mới là chuyện đơn giản	28
Phần cuối – Lời khuyên của bản thân Hoàng	31
Viết và chia sẻ code online với fiddle.....	34
Callback trong javascript.....	39
Series C# hay ho: Những điều thú vị trong C# (Phần 1).....	42
Series C# hay ho: Những điều thú vị trong C# (Phần 2).....	47
Series C# hay ho: Callback trong C# – Delegate, Action, Predicate, Func	53
Series C# hay ho: Lambda Expression.....	57
Series C# hay ho: Generic là cái thứ chi chi	61
Series C# hay ho: IEnumerable và yield, tưởng đơn giản mà lầm thứ phải bàn.....	65
Series C# hay ho: LINQ – Lột mặt nạ sự “bá đạo” của LINQ	69
Series C# hay ho – Tổng quan về Windows Services (WS).....	72
Windows Services là gì?.....	72
Tại sao phải sử dụng WS	73
Viết một Window Services đơn giản	73
Cài đặt Windows Service.....	79
Series C# hay ho: Giới thiệu Humanizer – Một thư viện khá thú vị của C#	83
1. Chuyển tên hàm, tên biến thành chuỗi có nghĩa	83
2. Thu gọn chuỗi	84
3. Chuyển enum thành chuỗi có nghĩa	84
4. Xử lý DateTime	84
5. Xử lý TimeSpan	84
6. Chuyển số thành chữ (Chỉ tiếng Anh).....	85

7. Xử lý file size (Khá hay).....	85
Series C# hay ho: C# 6 có gì hay ho nào.....	86
1. Hàm <i>nameof()</i>	86
2. Khai báo giá trị mặc định cho property.....	86
3. Khởi tạo danh sách với index.....	87
5. Dùng lambda expression để khai báo property và method	88
6. Format string ngắn gọn	89
7. Toán tử điều kiện null (?).	89
Series C# hay ho: So sánh 2 object trong C# (Deep compare).....	91
Series C# hay ho: C# đã tiến hóa như thế nào (Những thay đổi của C# từ 1.0 cho tới 5.0).....	98
1. C# 2 – Giải quyết vấn đề từ C# 1	98
2. C# 3 – Cải tiến trong data access, LINQ chào đời	99
3. C# 4 – Thay đổi nho nhỏ.....	101
4. C# 5 – Giảm thiểu đau đớn khi code bất đồng bộ	102
C# là ngôn ngữ tuyệt vời nhất. Java, PHP, C, C++, Ruby chỉ toàn là thứ rẻ tiền	103
Chúng ta đang xem ngôn ngữ lập trình như một thứ tôn giáo	103
Review sách: The Passionate Programmer – Những điều giúp developer phát triển sự nghiệp	107
Sơ lược nội dung sách	109
Nhận xét	110
Tạo dummy data với Faker và Mockaroo – Xa rời những ngày nhập tay nhảm chán	112
SOLID là gì – Áp dụng các nguyên lý SOLID để trở thành lập trình viên code “cứng”	119
1. Single responsibility principle	121
2. Open/closed principle	122
3. Liskov Substitution Principle.....	123
4. Interface Segregation Principle.....	124
5. Dependency inversion principle.....	125
Cột mốc 1000 rep trên stackoverflow – Kho khoe thành quả và chia sẻ kinh nghiệm.....	127
Phần 1 – Kho khoe	127
Phần 2 – Chia sẻ	129
Viết unit test cho javascript với Jasmine.....	133
1. Nhắc lại sơ về Unit Test.....	133
2. Giới thiệu về Jasmine.....	134
3. Demo Jasmine	135

Viết unit test cho javascript với Jasmine – Phần 2.....	140
1. Một số matcher của Jasmine	140
2. Cách dùng các hàm before, after.....	141
3. Sử dụng spy và mock	142
Áp dụng LINQ trong javascript, chuyện nhiều người chưa biết.....	145
Review sách: Clean Code: A Handbook of Agile Software Craftsmanship.....	149
Giới thiệu	149
Bài học rút ra.....	150
Lập trình viên “trình cao” thì nên đọc sách gì? – Phần 1	152
Lập trình viên “trình cao” thì nên đọc sách gì? – Phần 2	157
Tăng sức mạnh cho javascript với lodash	161
1. Giới thiệu tổng quan về lodash	161
2. Demo – 1 số function của string và function linh tinh.....	161
3. Một số function của object.....	164
4. Một số function của array	165
70 điều các developer giỏi thuộc nằm lòng – Phần 1	169
Hãy biết nói KHÔNG.....	172
Éo ai quan tâm đến code bạn viết đâu	176
Code chỉ là công cụ.....	177
Tập trung vào chức năng.....	177
Máy cái library vô dụng	178
Đúng, nhưng mà	179
Thiết kế của một trang web trở nên “bánh chành” như thế nào	180
11 điều luật mà mọi lập trình viên nên tuân theo	185
Cột mốc 1000 view – Lời cảm ơn chân thành đến bạn đọc và mọi người.....	190
Top 5 blog về IT đáng đọc	192
1. Joel on Software.....	192
2. Coding Horror	193
4. Jon Skeet (C# only).....	194
Một số mẫu comic hài hước về ngành IT – phần 1.....	196
Một số mẫu comic hài hước về ngành IT – phần 2.....	199
Cách tiếp cận 1 ngôn ngữ/công nghệ mới – Phần 1	206
1. Nền tảng (Fundamentals)	207

2. Kiến thức (Information)	208
3. Kỹ năng (Skills)	208
4. Thuần thục (Innovation).....	209
Cách tiếp cận 1 ngôn ngữ/công nghệ mới – Phần 2	210
Tìm tài liệu học – Giai đoạn sơ khởi.....	211
Bắt đầu học – Giai đoạn nhập môn	211
Áp dụng kiến thức – Giai đoạn nâng cao.....	212
Con đường phát triển sự nghiệp (Career path) cho developer	214
Fresher/Junior Developer.....	214
Developer	215
Quản lý hay kỹ thuật?	216
Những điều trường đại học không dạy bạn – Phần 1	217
Cách đọc và viết code	218
Sử dụng IDE, debug.....	219
Testing, unit test.....	220
Agile Development	221
Source code control system.....	221
Cách dùng thư viện và framework	222
Những điều trường đại học không dạy bạn – Phần 2	223
Tìm hiểu thị trường	224
Mở rộng quan hệ	224
Đầu tư vào bản thân	225
Xác định hướng đi.....	226
Phương pháp tự học, tự tìm câu trả lời.....	226
Những điều trường đại học không dạy bạn – Phần 3	228
Làm sao để phỏng vấn và xin việc	228
Văn hóa ứng xử nơi công sở	229
Những phương pháp thương thảo	230
Cách xây dựng tiếng tăm (reputation).....	231
70 điều các developer giỏi thuộc nằm lòng – Phần 1	234
70 điều các developer giỏi thuộc nằm lòng – Phần 2	238
[Tutorial] Trích xuất thông tin từ website với HTML Agilitity Pack.....	241
Trải lòng với bài viết thứ 50 – Cảm ơn sự ủng hộ của mọi người.....	248

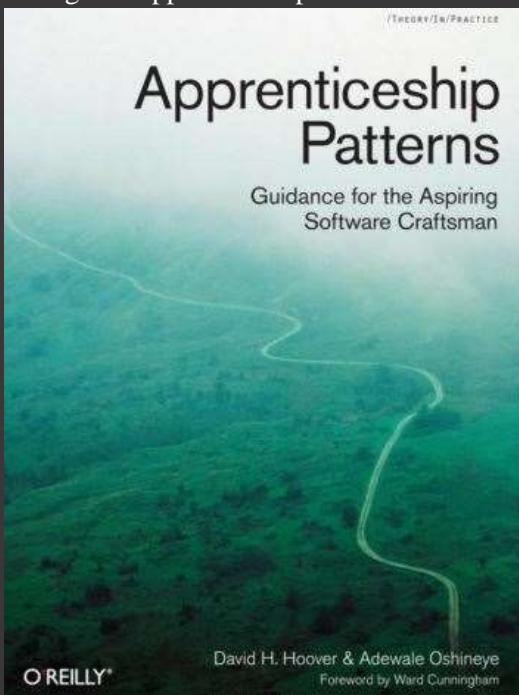
Muôn nẻo đường tìm việc – Phần 2: Vượt qua kì phỏng vấn like a boss	252
Những việc bạn cần làm trước khi đi PV	252
Quy trình phỏng vấn	253
Phone interview hoặc entry test	253
Face interview – Vòng quan trọng nhất	254
Muôn nẻo đường tìm việc – Phần 1: Viết CV rõ ràng và chuyên nghiệp	256
Tìm việc	256
Viết CV + nộp đơn xin việc	257
Một số câu phỏng vấn thú vị về lập trình	260
Luận về comment code (Phong cách kiểm hiệp)	265
Giai đoạn trẻ trâu đầy nhiệt huyết (Code không comment hoặc comment lung tung).	267
Giai đoạn tạm chín chắn trong suy nghĩ (Biết cách comment)	268
Cảnh giới vô kiểm thăng hưu kiểm (Vô comment thăng hưu comment)	268
[Tutorial] Tạo ứng dụng chat với 50 dòng code, Firebase và AngularJS	270
Giải thích đơn giản về CI – Continuous Integration (Tích hợp liên tục)	277
[Tutorial] Viết Unit Test trong C# với NUnit	282
Giới thiệu tổng quan về Unit Test.....	282
Unit Test trong C# với NUnit	283
Top 6 “trường dạy code” cho các developer	289
Code Academy	289
Pluralsight	290
Code School	291
Tuts+	292
Udacity	292
Skillshare.....	293
Những kỹ năng cần có của một web developer	294
Kỹ năng front-end	294
Kỹ năng back-end	295
Kỹ năng phân tích thiết kế	296
Sự khác biệt giữa Web Site và Web Application.....	298
1. Khái niệm website.....	298
2. Khái niệm webapp	298
3. So sánh website và web app.....	299

Tạo động lực học tập và làm việc – Sức mạnh của thói quen.....	301
[Tutorial] Hướng dẫn tích hợp Visual Studio với Github	306
NoSQL có gì hay ho – Tổng quan về NoSQL – Phần 1	319
Trước NoSQL, từng có một thứ gọi là ... SQL	319
Sau SQL, có một thứ gọi là ... NoSQL	320
Trong tương lai, NoSQL sẽ ... thế chỗ SQL ???	322
NoSQL có gì hay ho – Tổng quan về NoSQL – Phần 2	323
Key-Value Database	326
Document Database	326
Column-Family Database	327
Graph Database.....	327
Mapping thuật ngữ trong Relational Database và NoSQL database.....	328
Review sách: Presentation Zen – 99% sinh viên/lập trình viên đã làm slide tệ hại như thế nào	331
Giới thiệu nội dung	332
Nhận xét	340
Tại sao giới lập trình thù ghét Internet Explorer?.....	341
Ngày xưa ngày xưa, có 1 trình duyệt tuyệt vời tên là IE	342
Khi Microsoft ngủ quên trên chiến thắng	342
Lý do dân lập trình thù ghét IE	344
Tương lai nào cho IE.....	345
Hướng dẫn tích hợp Travis-CI vào project trên github	347
Tạm biệt ASWIG – Đôi dòng tâm sự của chàng junior developer – Phần 1	355
Cuối tháng 9/2014 – Bị phỏng vấn như super junior	355
Đầu tháng 10/2015 – Chập chững vào công ty, bị ma cũ “ăn hiếp”	357
Giữa tháng 4/2015 – Company trip, Thái Lan vẫn gọi	358
Đầu tháng 5/2015 – Buồn thay chuyện kể ở người đi.....	359
Tạm biệt ASWIG – Đôi dòng tâm sự của chàng junior developer – Phần 2	361
Đầu tháng 8/2015 – Nghỉ việc, tăng lương và cách xử sự của người lãnh đạo.....	362
Tổng quan về lập trình ứng dụng di động – Phần 1	366
Android – Kẻ chiếm miếng bánh lớn nhất	367
iOS – Vị vua không ngai.....	368
Windows Phone – Kẻ sinh sau đẻ muộn.....	369
Tổng quan về lập trình ứng dụng di động – Phần 2	370

Web App	371
Native App	372
Hybrid App	373
Tạm biệt Việt Nam – Chia sẻ kinh nghiệm nộp đơn du học – Phần 1.....	376
1. Chuẩn bị hồ sơ	377
2. Chọn trường và tìm ngành học.....	377
3. Nộp hồ sơ.....	380
Tạm biệt Việt Nam – Chia sẻ kinh nghiệm nộp đơn du học – Phần 2.....	382
4. Chờ offer letter, đóng tiền đặt cọc	382
5. Lấy CAS, xin Visa	383
6. Mua vé máy bay và lên đường	387
Ngồi xuống và viết blog đi nào.....	388
Viết blog có được lợi lộc gì không?.....	388
Thời gian rảnh rỗi đâu mà viết.....	389
Mình có biết gì đâu mà viết	390
Nghe hay đấy, chỉ mình cách bắt đầu đi nào.....	391
Chuyện về các cây đa cây đề trong làng Software Engineering.....	393
1. Dennis Ritchie.....	393
2. Uncle Bob	394
3. Martin Fowler	394
4. Joel on Software.....	395
Sự “bá đạo” của Chrome Developer Tools – Phần 1	396
1. Elements.....	397
2. Networks	397
3. Sources	398
4. Timeline & Profile	398
5. Resources	399
6. Audits	399
7. Console	400
Sự “bá đạo” của Chrome Developer Tools – Phần 2	401
1. Mở nhanh file javascript	401
2. Tìm kiếm trong source code.....	401
3. Select các element trong HTML	402

5. Chuyển đổi trạng thái của các element (hover, active)	403
6. Chính sửa nhanh nội dung các element.....	404
7. Tìm những event được bind vào một element	404
8. In object ra console nhò log và table.....	405
9. Lưu trữ console log, không mất log khi refresh lại trang.....	406
10. Lấy màu của các element khác trên trang web	407
Đôi dòng thông báo và cáo lỗi, kể lể cuộc sống ở UK	408
Review sách: Microsoft .NET – Architecting Applications for the Enterprise (1st Edition)	413
Giới thiệu nội dung	414
Nhận xét	415
Kết luận	416
Series C# hay ho: EPPlus – Thư viện Excel “bá đạo” – Phần 2	417
Thêm formula vào các ô trong Excel.....	417
Đọc nội dung từ file Excel	417
Series C# hay ho: EPPlus – Thư viện Excel “bá đạo” – Phần 1	422
Tạo File Excel.....	423
Export file Excel	424
Tút lại nhan sắc cho file Excel	426
Mặt tối của ngành công nghiệp IT – Phần 2	430
4. Giới technical không chưa đủ	430
5. Cấp trên thường “đeo biết gì cá”.....	431
6. Lập trình viên dàn đang mất giá.....	431
Kết. Thế giờ em phải sống nàm thao?.....	433
Mặt tối của ngành công nghiệp IT – Phần 1	435
2. Hackathon và Stackoverflow không “tốt đẹp” như bạn nghĩ.....	438
3. Vấn đề outsource và bóc lột.....	440
Dependency Injection và Inversion of Control – Phần 3: DI Container. Áp dụng DI vào ASP.NET MVC...	442
1. Tự viết 1 DI Container đơn giản	442
2. Sử dụng DI Container từ các framework có sẵn	447
3. Áp dụng IoC vào project MVC.....	449
Dependency Injection và Inversion of Control – Phần 2: Áp dụng DI vào code	452
Dependency là gì?	452
Làm sao để hạn chế coupling giữa các class. Đã có Inversion of Control	454

Dependency Injection và Inversion of Control – Phần 1: Định nghĩa	463
Nhắc lại kiến thức	464
Định nghĩa và khái niệm DI.....	465
Các dạng DI	467
Ưu điểm và khuyết điểm của DI	468
Series C# hay ho: Tại sao WinForm vẫn “chưa chết” – Có nên học WinForm hay không ?.....	469
WinForm, người bạn của tuổi thơ – đơn giản mà hiệu quả.....	469
Đừng nên đầu tư quá nhiều thời gian/công sức vào WinForm	470
Còn WebForm thì sao?	471
Review sách: The Healthy Programmer – Giữ sức khỏe để code và ăn chơi phè phỡn	473
Nhận xét	476
Mỗi tháng một cuốn sách – Những sách hay mình đã đọc trong năm 2015 – Phần 2	478
Tháng 7 – Apprenticeship Patterns – Guidance for the Aspiring Software Craftsman	

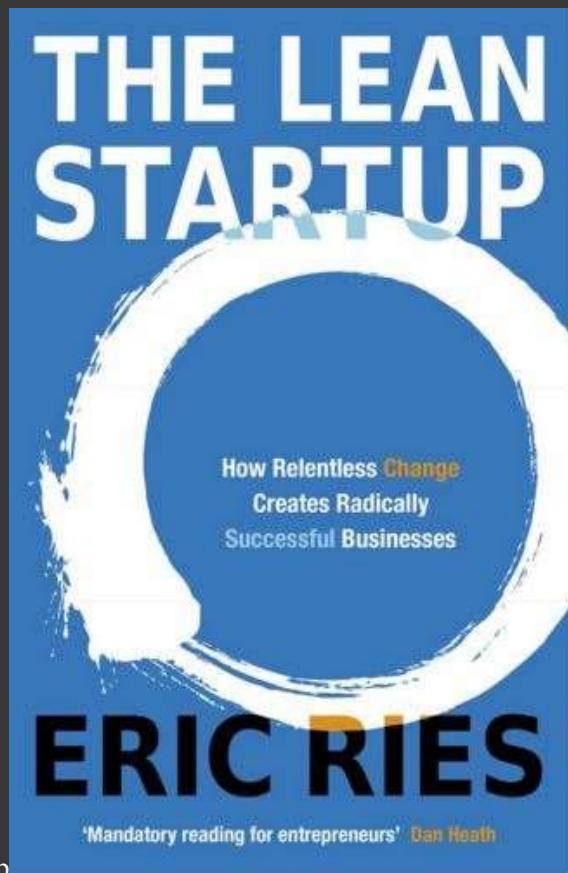


478



Tháng 8 – Suối nguồn

.....479



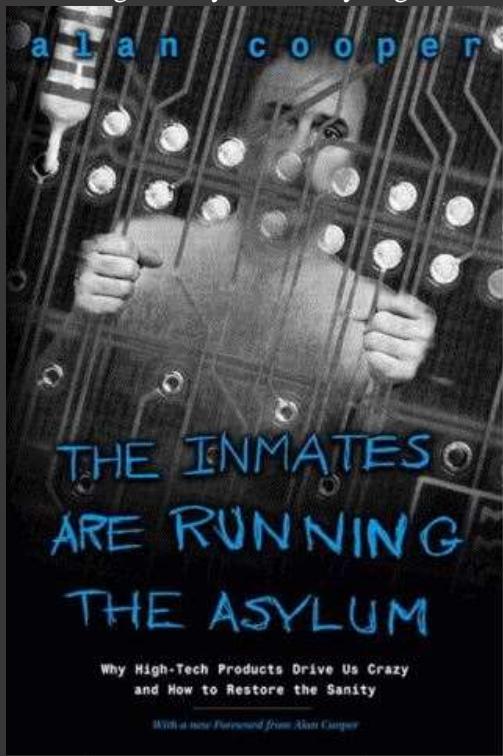
Tháng 9 – The Lean Startup

.....480



Tháng 10 – Kẻ thành công phải biết lắng nghe 481

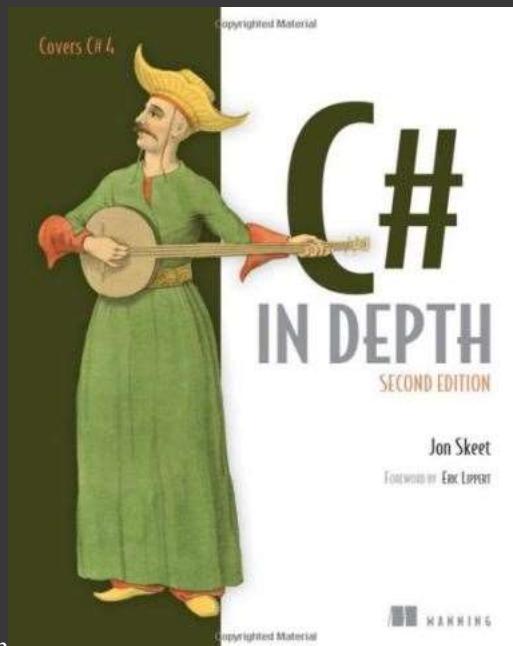
Tháng 11 – The Inmates are Running the Asylum – Why High-Tech Products Drive Us Crazy and



How to Restore the Sanity 482

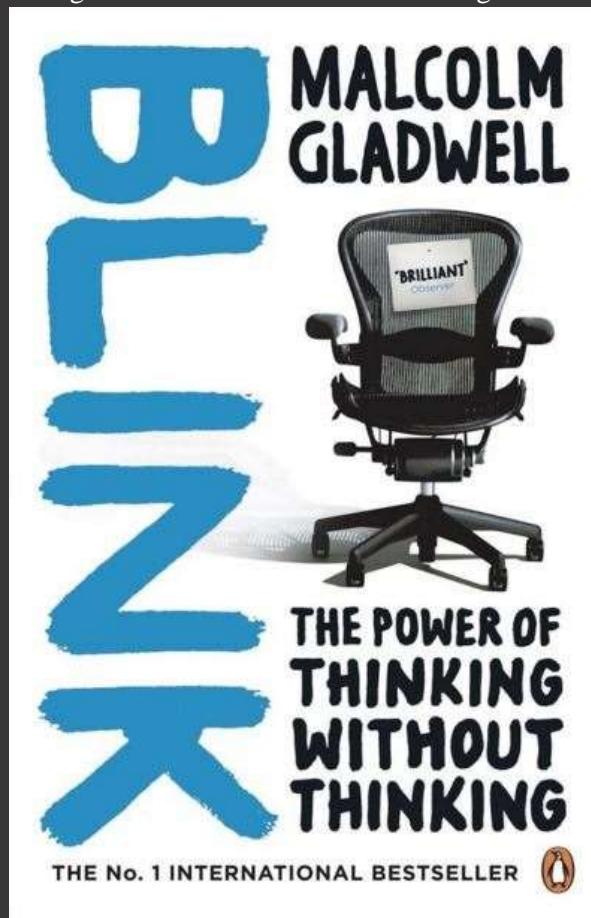
Mỗi tháng một cuốn sách – Những sách hay mình đã đọc trong năm 2015 – Phần 1 486

Tháng 1 – Zero to One: Notes on Startups, or How to Build the Future 486



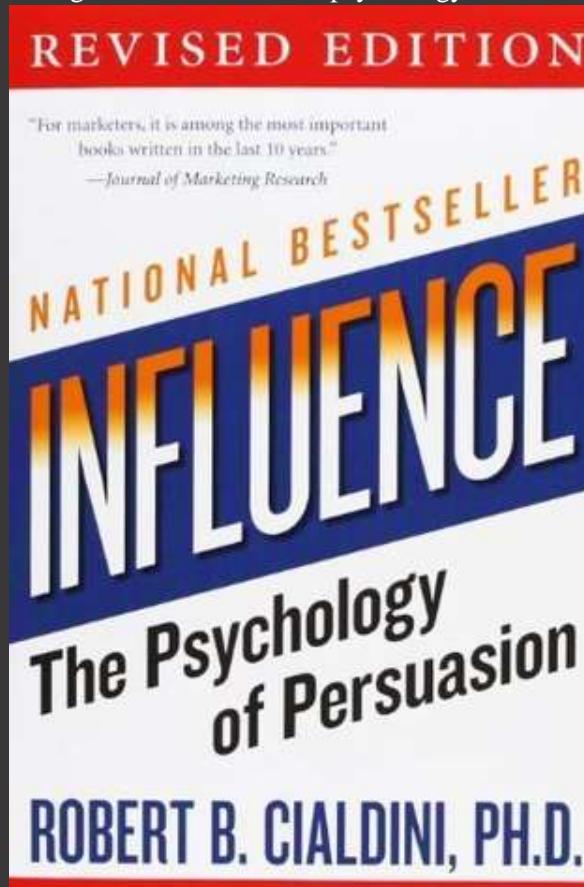
Tháng 2 – C# in Depth 3rd Edition 487

Tháng 3 – Blink: The Power of Thinking Without Thinking



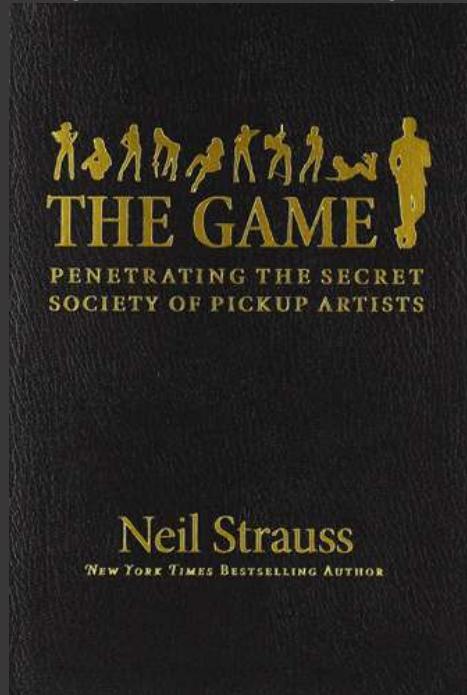
488

Tháng 4 – Influence – The psychology of Persuasion

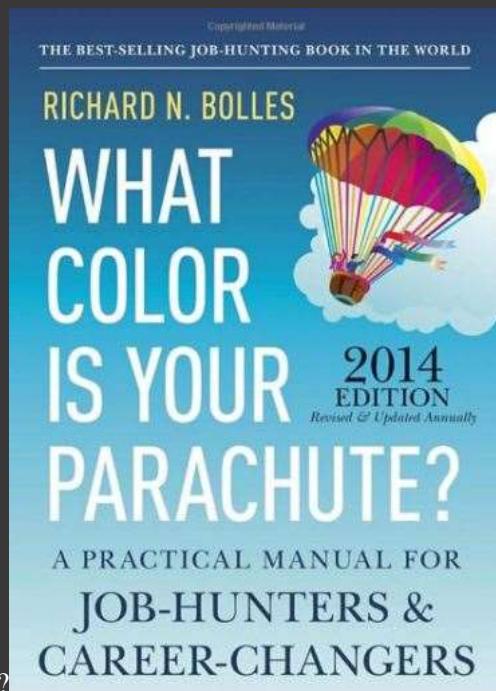


.....489

Tháng 5 –The Game: Penetrating the Secret Society of Pickup Artists



.....490



Tháng 6 – What color is your parachute?	491
Những mánh khốé “không bao giờ tiết lộ” của các lập trình viên vĩ đại	494
Mánh khốé code và test.....	494
Mánh khốé làm việc	496
Mánh khốé phát triển bản thân.....	497
Lỗ hổng bảo mật khủng khiếp của Lotte Cinema (Lưu trữ mật khẩu người dùng – Tưởng dễ mà không đơn giản)	498
Đăng nhập hả? Chỉ cần một bảng User, hai cột Username và Password là xong	498
Vậy mã hóa là được chứ gì, lầm trò!!	499
Ói giời phức tạp thế, cùng lầm thì lộ password trên trang của mình thôi mà	501
Nói đi nói lại một hồi, cũng đến cái “Lỗ hổng bảo mật khủng khiếp của Lotte Cinema”	502
Thực trạng học lập trình của một số thanh niên hiện nay.....	503
Thực trạng học lập trình của các “sinh dzién”	503
Lời kết.....	506
Giới thiệu tổng quát về Meteor	507
Meteor là gì?	507
Có nên thử sức với Meteor?	510
Một số kĩ thuật design cơ bản cho developer	512
Màu sắc	513
Font chữ (Typography)	514
Layout	515

Series JavaScript sida – Luận bàn về cái đít (this) trong javascript	517
Từ khi đít (this) nhỏ còn hiền lành vô hại	517
Tới khi đít lớn và gây bao rắc rối	518
Sự thật đắng lòng: Đôi khi cắm đầu ngồi CODE là cách ... ngu nhất để giải quyết vấn đề	524
Developer cũng nên học ... marketing – Chiến thuật bán hàng thú vị của John Sonmez.....	531
Từ một email úp úp mở mở	532
Thế rồi email thứ hai vào ngày hôm sau	533
Email cuối cùng – Em đã bị lừa, bị dính đòn quảng cáo.....	534
[Tutorial] Viết ứng dụng di động một cách dễ dàng với Ionic Framework	537
1. Cài đặt Ionic	538
2. Tạo ứng dụng đầu tiên	539
3. Chính sửa ứng dụng	540
4. Chạy ứng dụng trên di động	540
Một button trị giá 300 triệu đô – Cái nhìn khác về UI và chức năng	543
Ngày xưa ngày xưa, có một trang web bán hàng.....	543
Lũ lập trình viên luôn nghĩ rằng mình hiểu người dùng	544
Một cái nhìn khác về UI/UX và chức năng.....	545
Nhìn lại năm 2015, mừng blog thêm tuổi mới.....	547
12 tháng – Những chuyện ngày xưa ngày xưa	547
Cùng nhìn lại những cái nhất của blog.....	548
Định hướng cho năm 2016.....	549
Chuyện học tiếng Anh – Phần 1: Tôi đã học tiếng Anh như thế nào	550
Cuối năm cấp 3	552
Lên đại học.....	553
Lời kết	555
Chuyện học tiếng Anh – Phần 2: Tôi đã đạt TOEIC 945 như thế nào	556
Chuẩn bị tinh thần và tài liệu	557
Ôn thi TOEIC.....	558
Lời kết	559
Chuyện học tiếng Anh – Phần 3: Tôi đã đạt IELTS 7.5 như thế nào	561
Sơ lược về IELTS	561
Chuẩn bị tinh thần và tài liệu	562
Lời kết	566

Chuyện đầu năm – Lần đầu đi phỏng vấn xin việc nơi đất khách quê người (Phần 1).....	567
Duyên trời đưa đẩy.....	567
UX Developer là cái chi chi???.....	568
Nộp CV và chờ mòn mỏi	569
Chuyện đầu năm – Lần đầu đi phỏng vấn xin việc nơi đất khách quê người (Phần 2).....	571
Diện đồ nghiêm túc, lên đường.....	571
Phỏng vấn gian truân như đánh đố.....	572
Kết quả.....	575
Phải làm sao khi viết blog mà ... bí đề tài ???	576
Những điều nên làm khi bí đề tài	576
Những thói quen cần có để duy trì blog	578
Series Javascript sida – Pờ rồ tô tai (Prototype) là cái gì	580
Prototype là cái đếu gì?.....	580
Prototype dùng để làm gì?	582
Series JavaScript sida – Object trong JavaScript	585
Object là khỉ gì?	586
Khởi tạo object thế éo nào	587
Nghịch ngợm với object.....	589
Series Javascript sida – OOP trong JavaScript.....	593
OOP trong Java	594
OOP trong JavaScript	595
Kết luận	598
Series JavaScript sida – Bind, Call và Apply trong JavaScript	599
Trói đít (this) lại bằng bind	599
Call và Apply, tuy 2 mà 1, thấy 1 mà 2	601
Nhập môn Design Pattern (Phong cách kiểm hiệp)	607
Nhập đê	607
Hỏi thé gian DS là chi, mà bọn Dev thé nguyên sống chét.....	608
Design Pattern Kiếm Phố.....	610
Khẩu quyết nhập môn Design Pattern.....	611
Thay lời kết	612

About me

Về bản thân

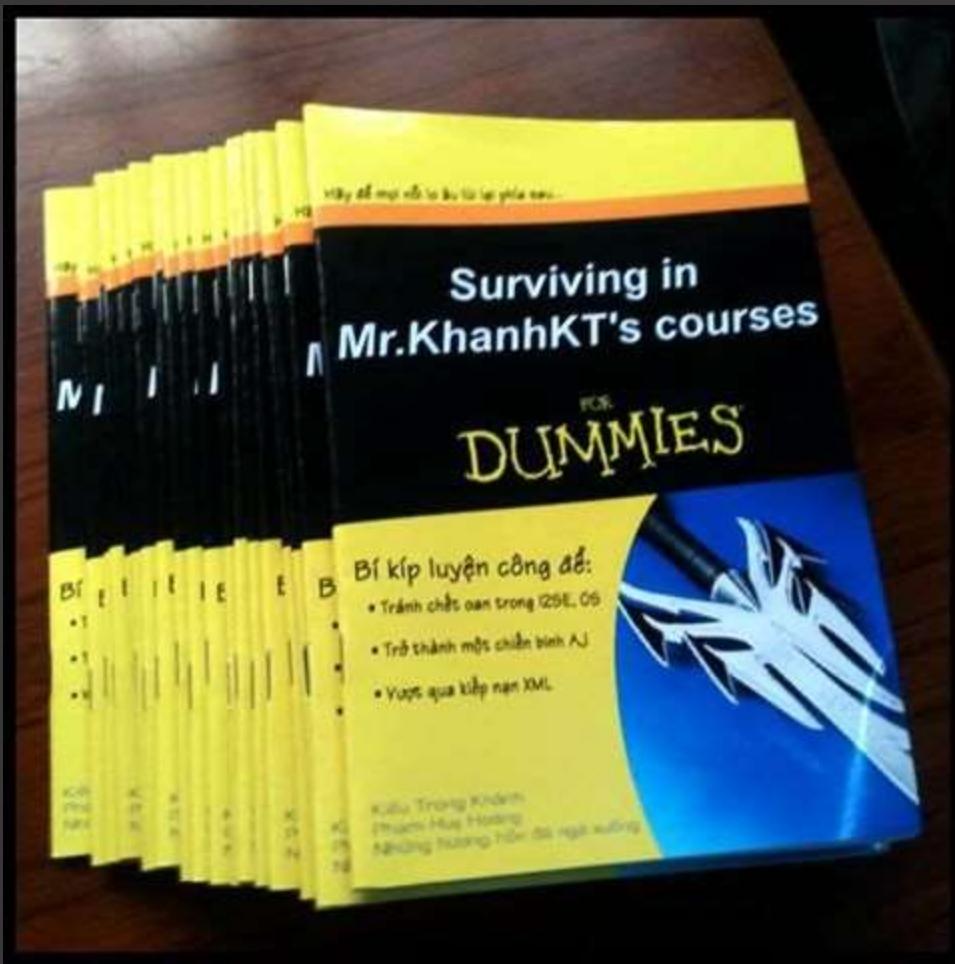
Tôi là **Phạm Huy Hoàng**.

Sinh ra trong một gia đình nho giáo lâu đời, có truyền thống code dạo, cài win dạo, do đó tên blog tôi chọn là “Tôi Đi Code Dạo” (Đùa đây).

Tôi là một **developer** (Thật ra là junior thôi, vừa ra trường ko lâu). Cái bạn đẹp giai cao cao ngoài cùng bên phải trong ảnh dưới ấy.



Ngoài ra tôi còn là một **blogger** (Chức danh tự phong, vì đây là blog đầu tiên tôi viết) Một cuốn sách tôi đã từng viết (Chỉ in 30 bản, xuất bản nội bộ trong Đại Học FPT, lợi nhuận tổng cộng 120k, đc chút tiền còn ăn sáng).



Khoe một chút, sau một quá trình cày cuốc lâu dài, cuối cùng tôi đã đạt mốc 1000 reputation trên stackoverflow



Về blog này

Đây là một blog về IT, mà cũng ko phải về IT.

Nội dung blog 50% vẫn sẽ là kĩ thuật lập trình, 50% còn lại là những kĩ năng mềm developer cần biết : Cách deal lương, sắp xếp thời gian, kĩ năng mềm, ngôn ngữ nền học, con đường phát triển nghề nghiệp... Đó là những điều quan trọng ko thua gì kĩ năng lập trình. Đó là phương châm của blog: Developer cần biết nhiều điều hơn ngoài code.

Chuyên sâu của tôi là C#.NET, do đó phần nhiều các bài viết lập trình sẽ thiên về .NET. Tuy nhiên những bài biết về web, design pattern, testing v...v ko phân biệt ngôn ngữ, ai cũng có thể đọc.

Tôi viết blog này với mục đích nâng cao khả năng diễn đạt, trình bày vấn đề, cũng như chia sẻ kinh nghiệm và kiến thức cho mọi người. Mọi comment góp ý hay ném đá, miễn là ko mang tính xúc phạm, đều được hoan nghênh.



Hỏi đáp

Dào, mới junior developer. đã làm được 1 năm, đã biết cái gì mà dạy người khác?
Kiến thức thu thập dc có thể thông qua kinh nghiệm hoặc sách vở, ko nhất thiết cứ làm lâu năm thì sẽ biết nhiều. Tôi đọc khá nhiều sách (Series For Dummies, Head First, Clean Code, Getting Real, Humanware, The Mithical Man-Month...), trong đó có nhiều thứ khá hay, rất đáng để chia sẻ.

Một góp ý nữa, tôi ko “dạy”, vì tôi vẫn chỉ là junior developer, hướng dẫn cho người khác cũng là một quá trình học, cho mình giỏi hơn. Do đó tôi rất hoan nghênh các developer khác vào “dạy” tôi và mọi người.

Dài quá ko muốn đọc, nhưng mà cái blog này viết về cái gì thế?
Tin 18+, lá cải, lừa tình, girl xinh, XXX, đùa đây. Nói ngắn gọn, blog này sẽ là nơi chứa những bài viết mà, developer nào đọc xong cũng sẽ vỡ òa : Mẹ, giá hồi xưa đi code mình biết cái này :))).

Cậu là lập trình viên, viết blog “dạy” người khác mà ko code dc 1 cái blog ra hồn à, sao lại phải dùng wordpress thế?

Tại sao phải viết lại từ đầu 1 blog engine, sau đó tìm host, deploy lên mạng, trong khi ta có thể tạo 1 blog đơn giản, theme có sẵn, plug-in chỉ với 3 bước? Sử dụng wordpress, tôi có thể tập trung vào việc viết blog, thay vì cố gắng code 1 engine mới lại từ đầu, rồi bỏ dở giữa chừng.

Cậu có bị ảnh hưởng bởi IT blogger nào ko?

Ý tưởng viết blog của tôi bắt nguồn từ John Sonmez, chủ nhân blog IT simple programmer. Đây là 1 bước tăng danh tiếng cũng như kỹ năng viết của bản thân

<http://simpleprogrammer.com/about-me/>

Ngoài ra, tôi còn đọc Joel On Software, ông là 1 lập trình viên nổi tiếng, có ảnh hưởng lớn, cũng nhờ >1000 bài viết về IT trên blog.

<http://www.joelonsoftware.com/>

Anh giỏi quá, tốt bụng quá quá, em muốn biết thông tin thêm để liên lạc với anh blah blah....

Địa chỉ mail của mình là: huyhoang8a5@gmail.com.

Facebook: <https://www.facebook.com/huyhoang.pham.106>

Rất khuyến khích các bạn developer nữ gửi thư làm quen, nhờ tư vấn v...v. Trường hợp developer nam hỏi code, làm quen, hay nhờ làm bài tập/đồ án giúp thì mình ko tiếp, xin cảm ơn :)



Short-circuit evaluation : Đoán mạch và chập điện

Posted on 29/01/2015 by Phạm Huy Hoàng

Bài viết hôm nay sẽ nhắc lại khái niệm “đoán mạch” của biểu thức logic trong lập trình. Mình dùng từ “nhắc lại” chứ không dùng từ “giới thiệu”, vì đây là điều ai cũng được học trong quá trình học, và 90% đều không áp dụng trong quá trình lập trình. (Số liệu thật đấy nhé, không chém đâu).

Trước tiên, chúng ta cùng ôn lại một chút về 2 biểu thức logic cơ bản là AND(&&) và OR(||). Chúng ta đều biết **false && true = false, true || false = true**. Điều ngược lại:**true && false = false, false || true = true**. Về logic, điều này đúng 100%, tuy nhiên bạn hãy thử điều này trong lập trình.

Giả sử ta có 2 hàm dưới đây, 1 hàm trả ra true, 1 hàm trả ra false

```
1 static bool ReturnTrue()
2 {
3     Console.WriteLine("ReturnTrue");
4     return true;
5 }
6
7 static bool ReturnFalse()
8 {
9     Console.WriteLine("ReturnFalse");
10    return false;
11 }
```

Đó bạn, đoạn code 1,2 và 3,4 sau đây cho kết quả giống hay khác nhau:

```
1 if (ReturnTrue() && ReturnFalse()) { }
2 if (ReturnFalse() && ReturnTrue()) { }
3
4 if (ReturnTrue() || ReturnFalse()) { }
5 if (ReturnFalse() || ReturnTrue()) { }
```

Đọc thử code, bạn sẽ nhủ thầm: Mẹ, dễ thế cũng đố, ai chẳng biết $a \&\& b = b \&\& a$, $a \mid\mid b = b \mid\mid a$. Đương nhiên là giống nhau rồi. Hãy chạy thử chương trình, bạn sẽ trọn mắt há mồm khi thấy kết quả:

```
1 if (ReturnTrue() && ReturnFalse()) { } // Return True  Return False
2 if (ReturnFalse() && ReturnTrue()) { } // Return False
3
```

```
4 if (ReturnTrue() || ReturnFalse()) { } // Return True
5 if (ReturnFalse() || ReturnTrue()) { } // Return False Return True
```

Tới đây, nếu chưa biết về short-circuit, thê nào bạn cũng: clgt(Cần lời giải thích). Đây là lời giải thích của mình: **Trong 1 số ngôn ngữ lập trình (C#, Java, Python), biểu thức logic thường có tính đoán mạch (Short-circuit)**.

Nói đơn giản là, với biểu thức AND, khi chương trình tìm thấy 1 điều kiện false, biểu thức AND sẽ trả kết quả false, những điều kiện sau điều kiện false đó sẽ không được thực thi. **Ở dòng code thứ 2, ReturnTrue không chạy**. Tương tự, ở dòng code thứ 3, khi nhận được kết quả True, biểu thức OR sẽ trả kết quả true, do đó ReturnFalse ko chạy.

Đọc tới đây, bạn sẽ gật gù như ngãm ra điều gì đó, rồi tự nhủ: “Ồ, cái này hay ghê, giờ mới biết, nhưng mà cũng vô dụng thôi, có áp dụng vào code bao giờ đâu”. Trước tiên, xin mời bạn xem thử dòng code dưới đây. Ta nhận về một list, xử lý list đó:

```
1 List<string> list = GetList();
2 if (list != null)
3 {
4     if (list.Count() != 0)
5     {
6         //Xử lý list
7     }
8 }
```

Ta có thể thấy người viết đã xử lý khá cẩn thận, nếu chỉ dùng hàm list.Count sẽ có khả năng gây ra NullPointerException, do đó ta phải check list != null trước. Tuy nhiên, khi đã biết về tính đoán mạch của biểu thức, ta hoàn toàn có thể viết như sau mà không sợ chương trình bị lỗi:

```
1 List<string> list = GetList();
2 //Nếu list null, chương trình sẽ không chạy
3 //câu lệnh list.Count, do đó không gây lỗi
4 if (list != null && list.Count() != 0)
5 {
6     //Xử lý list
7 }
```

Một ví dụ khác, code trước và sau khi áp dụng short-circuit, áp dụng cho việc so sánh chuỗi.

```
1 string name = "Hoang cute";
2 string search = "cute";
3
```

```
4 //Code cũ
5 if (search != null)
6 {
7     if (name.ToLower().Contains(search.ToLower()))
8     {
9         Console.WriteLine("Found");
10    }
11 }
12
13 //Code mới
14 if (search != null && name.ToLower().Contains(search.ToLower()))
15 {
16     Console.WriteLine("Found");
17 }
```

Bài viết đến đây là kết thúc. Đây chỉ là một điều nhỏ nhặt, nhưng nó giúp code của bạn đẹp + dễ đọc hơn “một chút”. Một anh senior từng nói với mình rằng, **giữa coder giỏi và dở, đôi khi chỉ là thằng code giỏi rành nhiều cái “một chút” hơn mà thôi**, các bạn đừng nên xem thường nó nhé =))).

Học ngôn ngữ lập trình nào bây giờ (3 Phần)

Posted on 13/01/2015 by Phạm Huy Hoàng

Đây một câu hỏi mà mình thường nhận được từ các em sinh viên mới ra trường, mới vào đại học, hoặc chưa biết gì về lập trình: “Giờ mình nên học ngôn ngữ lập trình nào đây?”.

Nghe đơn giản, nhưng đây là 1 câu hỏi có độ khó khá cao, sánh ngang với câu “Em nên làm nghề gì, vào đại học nào ...” của các em học sinh cấp 3. Trong phạm vi bài viết này, mình sẽ đưa ra một câu trả lời, dựa theo ý kiến cá nhân.

Tóm tắt nội dung bài viết

1. **Trước khi hỏi câu này, hãy tự hỏi : Mình muốn học lập trình để làm gì?**
2. **Lựa chọn thật ra không quan trọng. Học một ngôn ngữ mới là chuyên đơn giản.**
3. **Lời khuyên của bản thân Hoàng**

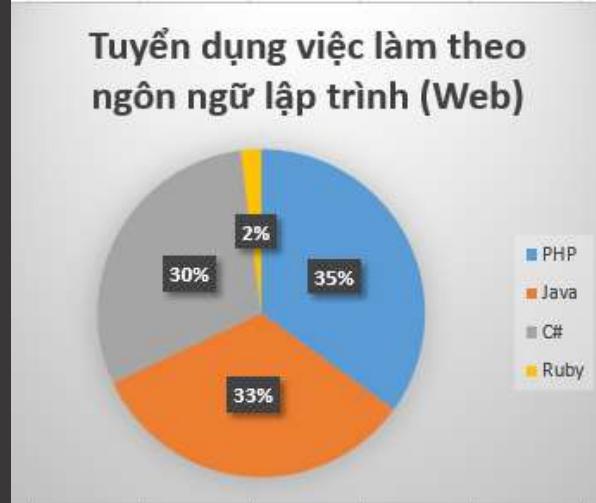
Phần 1 – Trước khi hỏi câu này, hãy tự hỏi : Mình muốn học lập trình để làm gì?

Khi được hỏi “Giờ mình nên học ngôn ngữ lập trình nào đây?”, mình luôn hỏi lại câu này “Bạn/Em muốn học lập trình để làm gì?”. Trả lời được câu hỏi này, bạn đã xác định được 50% ngôn ngữ mình cần học. Dưới đây là 1 số câu trả lời mình hay nhận được.

1. Em vừa ra trường, trường chỉ dạy C, C++, ... giờ em cần học ngôn ngữ gì để dễ kiếm việc làm, lương cao? Thị trường việc làm IT hiện tại rất nhiều, tạm chia làm 3 mảng: embedded, web và mobile.

- Mảng embedded: yêu cầu khá cao về trình độ, lập trình ngôn ngữ **C, C++**, có thể có Java. Nếu bạn là lập trình viên C++ cứng, mức lương rất cao, mức độ cạnh tranh cũng ko nhiều.
- Mảng mobile: Chiếm thị phần cao nhất vẫn là app cho Android viết bằng Java, tiếp theo là app cho IOS, viết bằng Objective-C. Java là một ngôn ngữ khá dễ học, độ phổ biến cũng cao, ứng dụng rộng. Nên học **Java** vì có thể chuyển qua các mảng khác khá dễ dàng.
- Mảng web: Các công ty outsource ở VN hiện tại đều tuyển LTV **C#.NET** và **Java**, do đó nhu cầu khá cao. Tuy nhiên nhu cầu cao nhất vẫn là **PHP**. Cần lưu ý 1 điều là số lượng developer PHP khá đông và hung hăn,

thượng vàng hạ cám cũng nhiều, do đó mức độ cạnh tranh khá cao. Như đã nói, số lượng công việc cần PHP rất đông, từ part-time, full-time đến free-lance, nếu giỏi PHP thì mức lương ko hè thấp nhé. Trường mình có 1 bạn SV năm 2, chỉ kiếm việc free-lance PHP trên freelancer.com cũng kiếm được 20 triệu/tháng.



=> Kết luận: Nếu muốn học để kiếm tiền, hãy xác định mình sẽ làm mảng công việc nào, sau đó chọn ngôn ngữ được yêu cầu nhiều. Hiện tại có 1 số ngôn ngữ như **Rails**, **Python**,... ít người học, developer giỏi ngôn ngữ này cũng có thu nhập khá (Vì hiếm nên quý =))).

2. Mình muốn làm 1 website, 1 ứng dụng cho người nhà, bản thân v....v. Có 1 số bạn học tài chính ngân hàng, kinh tế trả lời mình như vậy.

=> Nếu bạn muốn làm 1 ứng dụng di động, Java là lựa chọn tốt nhất. Còn việc tạo 1 website, hiện tại có rất nhiều hướng dẫn tạo website bằng Joomla, Drupal,... ko cần kiến thức lập trình. Các bạn có thể học thêm PHP để có thể tùy biến, thêm tính năng cho trang web.

Tới đây cũng đã dài, mình biết các bạn trẻ Việt Nam không muốn đọc dài nên sẽ viết ngắn gọn đủ ý nhất có thể. Hẹn gặp lại các bạn trong [phần 2: Lựa chọn thật ra không quan trọng. Học một ngôn ngữ mới là chuyện đơn giản.](#)

Phân 2 – Lựa chọn thật ra không quan trọng, học một ngôn ngữ mới là chuyện đơn giản

Đọc tới đây, có lẽ nhiều bạn sẽ quăng bom, ném gạch mình tới tấp “dám chắc thằng chủ thớt không phải coder, phán như thánh”.

Trước khi ném đá, mong các đồng chí bình tĩnh nghe Hoàng giải thích trình bày. Mình cũng từng là sinh viên IT như các bạn. Môn đầu tiên về lập trình mình học khi vào Đại Học là: “Cơ bản lập trình với C”. Mình từng điên đầu với khai báo biến, tách hàm, điều kiện, vòng lặp ,IO.... Môn tiếp theo là “Lập trình hướng đối tượng với C++”. Phải thú thật C++ không phải là ngôn ngữ phù hợp để học hướng đối tượng (Lẽ ra nên dùng Java hay C#). Mình từng nhầm lẫn trước các khái niệm “tính bao đóng, tính kế thừa”. Do đó, bản thân mình cũng biết sự khó khăn gấp phải khi học 1 ngôn ngữ. Tuy vậy, mình vẫn khẳng định **học một ngôn ngữ mới là chuyện đơn giản**.



Vì sao? Hãy tự xem lại kiến thức lập trình bạn có được khi vừa ra trường:

- Học qua 1,2 ngôn ngữ gì đó
- Cấu trúc dữ liệu và thuật toán
- Thiết kế, truy vấn cơ sở dữ liệu
- Design pattern (Có thể)
- Khả năng design front end

Khi mới tiếp cận lập trình, chúng ta cảm thấy khó khăn vì phải làm quen với vô số **khái niệm** mới. Tuy nhiên, khi đã có kiến thức cơ bản, việc tiếp cận ngôn ngữ mới trở nên **rất dễ dàng**. Bạn có thể tự hỏi, mình học gì khi học 1 ngôn ngữ mới? Đây là câu trả lời:

- Cách khai báo hàm, biến
- Cách khai báo vòng lặp, điều kiện if/else
- Các kiểu cấu trúc dữ liệu: list, set, tuple, ...

- IO, multi-thread, delegate, event
- IDE phù hợp, cách build, debug
- Các framework, cách sử dụng,

```

    package com.truhin.semantics.rest.publicAPI;
    import ...
    @Stateless
    @Path("/systemInfo")
    @PermitAll
    public class SystemInfo {
        @GET
        @Path("/{buildId}")
        @PermitAll
        public Response getVersion() throws URISyntaxException {
            File warfile = new File(this.getClass().getProtectionDomain().getCodeSource().getLocation().toURI());
            SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss z");
            simpleDateFormat.setTimeZone(TimeZone.getTimeZone("Europe/Moscow"));
            String result = simpleDateFormat.format(new Date(warfile.lastModified()));
            return Response.ok(result).build();
        }
        @GET
        @Path("/{username}")
        @PermitAll
        public Response getUserByUsername(String result) {
            if (securityContext.get() != null) {
                result = securityContext.get().notify();
            } else {
                result = securityContext.get().wait();
            }
            return Response.ok(result).build();
        }
        @GET
        @Path("/logout")
        @PermitAll
        public Response logout(@Context HttpServletRequest request) {
            request.getSession().invalidate();
            return Response.noContent().build();
        }
    }

```

Nếu bạn đã biết cách viết for, if/else, while ... trong Java, khi chuyển qua học C# hoặc javascript, cấu trúc hàm for, if/else... vẫn giữ nguyên. Kiến thức của bạn được kế thừa từ ngôn ngữ lập trình trước, do đó việc học sẽ diễn ra nhanh hơn. Hoặc khi bạn đã rõ cơ chế làm việc của ASP.NET RestAPI, việc học cách xây dựng RestAPI bằng Spring của Java cũng không quá khác biệt. Mình từng tự học Python mất 1 tuần, và học framework Django mất khoảng 2 tuần nữa. Lý do mình học nhanh vậy là vì:

- Mình đã có kiến thức cơ bản về lập trình (class, data structure)
- Mình biết những gì mình cần học. Khi mới lập trình, bạn không biết mình cần học gì. Tuy nhiên nếu đã có kiến thức nói chung về lập trình, bạn sẽ biết mình tập trung học những gì, điều này tiết kiệm rất nhiều thời gian.
- Mình biết là mình làm được. Khi mình hỏi bạn bè chung ngành “Học 1 ngôn ngữ mới mất bao lâu”, hầu hết đều trả lời “1 tháng hoặc hơn”. Vì thấy tồn tại nhiều thời gian + khó khăn như vậy nên hầu như họ rất “ngại” học ngôn ngữ mới.



Điều mình muốn nhắn nhủ với các bạn qua bài viết này: Đừng sợ mình sẽ chọn nhầm ngôn ngữ, cứ học đi. Việc học 1 ngôn ngữ mới khi bạn đó có kiến thức cơ sở khá đơn

giản, không hề khó khăn và mất thời gian như bạn nghĩ. Thêm vào đó, việc biết nhiều ngôn ngữ sẽ giúp bạn có lợi thế hơn khi xin việc =))

Hẹn gặp lại các bạn trong phần cuối: **Lời khuyên của bản thân Hoàng**

Phản cuối – Lời khuyên của bản thân Hoàng

Như tựa đề, dưới đây là một số lời khuyên của mình, dựa theo kinh nghiệm cá nhân (Mình chỉ có kinh nghiệm mảng web và mobile, nên các lời khuyên có thể sẽ không áp dụng được cho mảng embedded system).

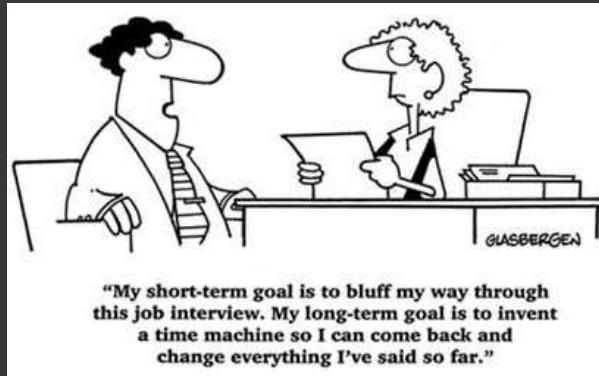
1. Đừng quá tập trung vào công nghệ. Đây là lời khuyên của thầy mình. Một số bạn rất ham hố tìm hiểu, học những công nghệ mới. Điều này không xấu, nó còn giúp bạn học được nhiều thứ mới, dễ xin việc. Công nghệ là thứ dễ thay đổi, mau hết thời. VD: Silverlight, công nghệ Web của Microsoft từng làm mưa làm gió, giờ đang ngắc ngoải. WinForm thì đã lỗi thời, ... thời gian bạn bỏ vào học các công nghệ này xem như lãng phí. Chưa kể, khi 1 công nghệ được nâng cấp, vd ASP.NET MVC 3 lên MVC 4, ta cần học thêm những điều mới trong bản 4. Việc đầu tư quá nhiều thời gian vào học công nghệ mới sẽ làm bạn dễ đuối, đôi khi là lãng phí thời gian.



2. Hãy đầu tư vào những thứ lâu bền. Nếu không tập trung vào công nghệ, chúng ta nên học gì? Đó là những thứ ít thay đổi, nhưng lại vô cùng cần thiết với developer:

- Kiến thức cơ bản. Đừng vội cười, mình biết nhiều bạn tuy đã ra trường nhưng vẫn còn lờ mờ về khái niệm pointer, delegate, multi-thread,... Đây là những kiến thức mà ta sẽ sử dụng trong suốt cuộc đời lập trình, và hầu như sẽ chẳng bao giờ thay đổi. Bỏ thời gian ra học kỹ chúng sẽ không hại gì, phải không?
- Cách viết code: Đặt tên biến như thế nào, tách hàm ra sao, comment như thế nào. Có 1 câu nói: LTV dở viết code cho máy hiểu, LTV giỏi viết code cho cả máy và người hiểu. Có nhiều bạn code xong, 6 tháng sau nhìn lại không hiểu mình viết gì. Khi đi làm, có nhiều giai đoạn bạn phải maintainance, đọc những đoạn code do “thánh” viết và chửi thề “Thằng này ngu thê”. Hãy code có lương tâm, nghĩ tới người sau này sẽ đọc, fix bug, sửa code của mình bạn nhé (Biết đâu người đọc lại là mình đây =)).

- Design Pattern. Nắm vững design pattern sẽ giúp bạn **ghi điểm khi phỏng vấn**. Đùa đà, nó sẽ giúp bạn giải quyết rất nhiều vấn đề thường gặp trong khi code (Sẽ nói rõ hơn về chuỗi bài design pattern sau này). Ngoài ra, design pattern sẽ giúp bạn học và hiểu 1 framework mới dễ dàng hơn, 1 framework tốt thường áp dụng khá nhiều design pattern.
- Một số mô hình thường dùng: MVC, MVVM, MVP, mô hình Client-Server ... Đây là câu hỏi được hỏi trong 90% các cuộc phỏng vấn, cũng là kiến thức cực kì quan trọng. Hiểu và nắm vững các mô hình trên sẽ làm cuộc đời lập trình của bạn thoải mái hơn nhiều.



3. Front end là bắt buộc. Không cần biết bạn học PHP, Java, C# ... bạn vẫn cần học và rành front end. Ở đây là **HTML, CSS, JS**. Trừ khi bạn làm ứng dụng mobile, embedded system, bạn sẽ phải đối mặt với html, css, js 8 tiếng làm việc mỗi ngày. Có gắng làm quen và làm thân với tụi nó nhé.

4. Lời khuyên cuối cùng: Nếu bạn quá dư thời gian, muốn thành thạo 1 ngôn ngữ nào đó, mình khuyên bạn nên chọn javascript (mà ko phải là PHP). Vì các lý do sau đây:



So sánh phũ phàng khi “Good parts” của js chỉ là một cuốn sách mỏng dính

- Javascript có ứng dụng rộng khắp, số lượng framework viết cho nó rất nhiều. Bạn có thể tìm hiểu về MEAN stack (MongoDB – Express – AngularJS –

NodeJS), xây dựng toàn bộ website từ front-end tới back-end sử dụng thuần javascript.

- Học javascript không cần cài đặt nhiều. Chỉ cần mở notepad, save lại dưới dạng html, mở bằng trình duyệt (hầu như máy tính nào cũng có), bạn đã có thể viết những dòng js đầu tiên. Học C#, Java ... sẽ phải cài 1 đồng IDE, SDK... rất lâu và mệt mỏi.
- Javascript là một ngôn ngữ hay và đẹp. Ngày xưa mình rất ghét ngôn ngữ này, và cộng đồng cũng có rất nhiều người ghét. Nó từng là con ác mộng của các developer. Tuy nhiên, các framework gần đây (Jquery, Angularjs,...) đã làm javascript có giá hơn, đẹp hơn, được nhiều người săn đón. Có thể nói javascript thuần như 1 con nhóc xấu xí ma chê quỷ hon, nhưng nhờ công nghệ trang điểm siêu ảo diệu (từ các framework), nên mới trở nên long lanh như thiên nga vậy.



Độ hot của Javascript, bây giờ và trước kia

Tổng kết lại những điều mình đã nói qua 3 bài viết cho bạn nào lười đọc

- Trước khi hỏi “Cần học ngôn ngữ gì”, hãy tự trả lời “Học lập trình để làm gì”.
- Đừng lo chọn sai ngôn ngữ, học một ngôn ngữ mới rất đơn giản.
- Đừng chạy theo công nghệ, hãy tập trung vào những thứ lâu bền
- Nếu quá dư thời gian, hãy tập trung vào học **javascript**.

Mình sẽ cố gắng cập nhật bài viết vào thứ 3 và thứ 5 mỗi tuần, các bạn nhớ theo dõi đón xem nhé.

Viết và chia sẻ code online với fiddle

Posted on 03/02/2015 by Phạm Huy Hoàng

Trước khi giới thiệu fiddle là gì, mình xin kể các bạn nghe 1 câu chuyện “có thật” sau. Một hôm nọ, khi mình đang ở công ty, bỗng nhận được mes của thằng bạn.

- Thằng bạn: È mày, cái css của tao bị sao áy, chỉnh lại giùm tao với. Cái jquery cũng ko chạy luôn, bấm nút xong nó ko ra gì hết. Team viewer giúp tao phát
- Mình: Máy tao cty ko cài team viewer được, quăng file qua đây.
- Thằng bạn: *Hì hục send file*
- Mình: *Mở file lên*, đọc mỗi 1 file html, ko css js gì sát, nhắn lại: Có mỗi file html, sao chạy?
- Thằng bạn: *Hì hục zip folder, send lại*
- Mình: *Hì hục tải về, giải nén. Fix lỗi, send lại*. Xong rồi đó.
- Thằng bạn: *Hì hục tải về, giải nén. Test* Ủa, mày sửa chỗ nào vậy
- Mình:



FACEPALM

Because describing how dumb that was in words just won't work.

Bài học rút ra từ câu chuyện này là gì? Thật ra là éo có bài học nào cả. Tuy nhiên, các bạn có thể thấy việc gửi file code qua lại rất mất công, tốn thời gian. Với fiddle, ta có thể chia sẻ code (html, css, js*) **nhanh chóng, tiện lợi mà không cần cài đặt bất kì thứ gì**.

Vậy fiddle là gì. Fiddle, hay còn gọi là code playground, là nơi cho phép bạn code, hiển thị kết quả online, sau đó send dưới dạng link để chia sẻ cho bạn bè. Một số fiddle nổi tiếng hiện nay: jsfiddle, codepen, jsbin, plunker, csharpfiddle... Link sau đây là 1 fiddle: <http://jsfiddle.net/blazeeboy/fNPvf/>

Như các bạn đã thấy, ta có cửa sổ HTML, CSS, JS và output. Chúng ta có thể thoải mái sửa code, bấm Run và xem thành quả. Sau đó, ta copy link này, gửi cho bạn bè 1 cách nhanh chóng. Dưới đây là những lý do chúng ta nên sử dụng fiddle:

- Bắt tay vào code nhanh chóng, chỉ cần vào link, ko cần phải mở notepad hay IDE nào khác.
- Hỗ trợ collaborator, 2 người có thể code chung trên 1 fiddle, việc update sẽ được diễn ra đồng thời. Rất phù hợp khi bạn muốn fix bug hoặc dạy kèm ai đó lập trình.
- Fiddle tự hỗ trợ revision, mỗi khi bạn lưu, fiddle sẽ tự động đánh version cho phiên bản đó. Do đó bạn có thể dễ dàng quay lại phiên bản trước nếu code sai.
- Fiddle có hỗ trợ các thư viện javascript nổi tiếng như Jquery, AngularJS, chỉ với 1,2 cú click đơn giản, thay vì chúng ta phải tải file javascript về máy, link với file html
- Fiddle được lưu trữ dưới dạng link. Khi demo code, ta có thể gắn kèm link fiddle, người đọc click vào link để xem demo, và có thể nghịch code để hiểu rõ hơn. VD như đây là demo show alert khi click 1 button: <http://jsbin.com/potivubifi/1/watch?html,js,output>

Mình sẽ hướng dẫn cách sử dụng 2 fiddle phổ biến nhất là [JSFiddle](#) và [JSBin](#). Các bạn click vào hình để xem ảnh lớn nhé.

The screenshot shows the JSFiddle interface with the following components:

- Top Bar:** Includes "Run", "Save", "TidyUp", "JSHint", "Collaboration", and "Login/Sign up".
- Frameworks & Extensions:** A dropdown menu showing "jQuery 2.1.0" selected, with other options like "jQuery Mobile 1.4.2", "Bootstrap 3.2.0", "AngularJS", and "Bootstrap 2.3.2".
- HTML Area:** Contains the following HTML code:

```
<button id="btn" type="submit">Hello</button>
```
- JavaScript Area:** Contains the following JavaScript code:

```
var btn = document.getElementById("btn");
btn.onclick = function() { alert("Hello"); };
```
- CSS Area:** Contains the text "CSS" in red.
- Result Area:** Contains the text "Kết quả hiện ra sau khi bấm Run" in red.
- Left Sidebar:** Includes sections for "Fiddle Options", "External Resources", "Languages", "Ajax Requests", and "Legal, Credits and Links".
- Bottom Bar:** Includes "Keyboard shortcuts" and a "Help" link.

Red annotations are present in several areas:

- "Chọn thư viện: jQuery,..." is highlighted in red.
- "Nhập HTML" is written in red above the HTML code area.
- "Nhập CSS" is written in red above the CSS area.
- "Nhập JavaScript" is written in red above the JavaScript code area.
- "Kết quả hiện ra sau khi bấm Run" is written in red above the result area.

Tổng quan về jsfiddle

How to show & hide line numbers
@import with Sass
Upgrade to pro now

New bin

Textarea editor mode

File • Add library Share HTML CSS JavaScript Console Output Login or Register Blog Help

HTML

Output only (with live reload)

`<!DOCTYPE html>
<html>
<head>
<script src="//code.jquery.com/jquery-1.11.3.min.js"></script>
<meta charset="utf-8">
<title>JS Bin</title>
</head>
<body>
<button id="btn" type="button">jQuery</button>
<button id="btn2" type="button">Hello from jQuery</button>
</body>
</html>`

Select panels to show: Link kết quả

JavaScript

`var btn = document.getElementById("btn");
btn.onclick = function() { alert("Hello"); };

$("#btn2").click(function(){ alert("Hello from jQuery");});`

Output

Run with JS Auto-run JS

Hello Hello from jQuery

Kết quả tự cập nhật khi ta sửa code, không có nút Run

Link

Embed

Link code + kết quả

Codecast

http://jsbin.com/potivubif/1/watch?output

Khóa, không cho người khác sửa

This bin is now locked from further changes

Tổng qua về jsbin

Không chỉ HTML, CSS, JS, hiện tại chúng ta đã có fiddle cho C# ở đây, các bạn có thể vào nghịch thử. <https://dotnetfiddle.net/> (Có hỗ trợ nhắc lệnh).

Fiddle còn một số khuyết điểm như: Chưa hỗ trợ nhắc lệnh, chỉ cho phép sử dụng 1 library (Trong jsfillde, với plunker hoặc jsbin ta có thể dùng nhiều library), Tuy nhiên, dễ thấy so với sự tiện lợi, nhanh chóng của chúng thì những khuyết điểm sau không hề quan trọng. Hãy sử dụng fiddle và dù dỗ bạn bè sử dụng nhé. Nếu cần hướng dẫn thì cứ link tới bài viết của mình này. Thân chào.

Callback trong javascript

Posted on 05/02/2015 by Pham Huy Hoàng

Callback là một khái niệm không mới. Tuy nhiên, nó là một trong những khái niệm khá lằng ngoằng và dễ nhầm lẫn trong lập trình. Với 1 số bạn có basic về C++, Java hay C#, ta thường biết đến callback qua khái niệm delegate (**con trỏ hàm**).

Bài viết đầu tiên, mình xin giới thiệu callback trong javascript. Lý do chọn javascript là vì Callback trong javascript là đơn giản, dễ hiểu nhất. Bài viết nhắm tới đối tượng là các bạn beginner nên mình sẽ cố gắng viết đơn giản nhất có thể.

1. Khái niệm về callback

Đầu tiên, xin nhắc lại đôi chút về khái niệm callback:

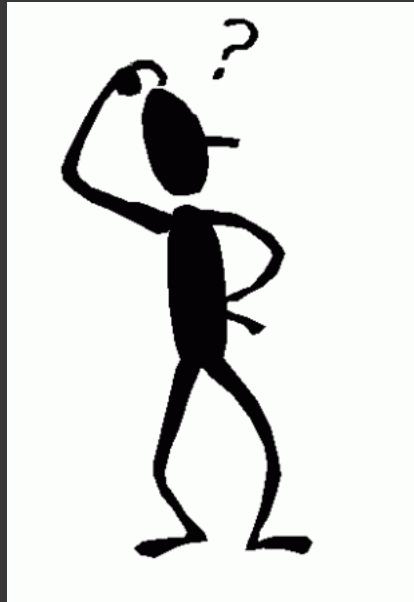
A callback is a piece of executable code that is passed as

an argument to other code, which is expected to call back

(execute) the argument at some convenient time.

Nói một cách dễ hiểu, callback tức là ta truyền một đoạn code (**Hàm A**) này vào một đoạn code khác (**Hàm B**). Tới một thời điểm nào đó, Hàm A sẽ được hàm B gọi lại (**callback**).

Tới đây các bạn vẫn chưa hiểu? Ok, ngày xưa khi nghe các giáo viên dạy cái này mình cũng ko hiểu gì cả =))). Mình xin đưa ra 1 ví dụ đơn giản dễ hiểu ở phần dưới.



2. Ví dụ của callback. Ứng dụng trong jQuery

Bạn có việc phải đi công tác xa nhà. Bạn dặn vợ, trong thời gian bạn đi, nếu như có ai giao quà tới, hãy đem qua tặng em gái dễ thương hàng xóm. **Hàm A** ở đây là việc tặng quà cho em hàng xóm:

```
1     function tangQua (qua) {  
2         return console.log("Đã tặng " + qua);  
3     }
```

Hàm B ở đây là việc vợ bạn ở nhà. Ta truyền hàm A vào như 1 argument cho hàm B, tại 1 thời điểm nào đó, **hàm B sẽ gọi hàm A** (Tức là vợ bạn mang quà qua cho em hàng xóm).

```
1     function oNha (vo, tangQua) {  
2         var qua = "Quà đã nhận";  
3         tangQua (qua);  
4     }
```

Như các bạn thấy, hàm A được truyền vào như 1 argument. Các bạn biết jQuery có thể sử dụng rất nhiều callback mà ko biết. Hãy xét ví dụ dưới đây, ta hiển thị 1 pop-up khi click vào 1 button. Bài viết trước nói về fiddle nên lần này mình sẽ đưa link fiddle làm ví dụ:

<http://jsbin.com/potivubifi/1/watch?html,js,output>

Ở đây, function **showPopup** chính là hàm A, còn `$('#btn').click()` là hàm B. Hàm B sẽ gọi hàm A khi ta click vào nút. Các bạn viết jQuery thường dùng cách ở dưới, nhưng **không hiểu bản chất của việc mình đang làm**.

3. Ứng dụng của callback

Như mình đã nói, Javascript là một ngôn ngữ khá thích hợp để giới thiệu callback. Ta có thể truyền thẳng 1 function vào 1 function khác, vì trong javascript 1 function được xem như 1 đối tượng (object). **Trong các ngôn ngữ khác như C#, ta phải dùng delegate để truyền 1 function vào function khác** (Nhớ kĩ câu này, mình sẽ nói rõ hơn ở bài sau).

Một số bạn sẽ hỏi: Callback mình code có xài mấy đâu? Thật ra trong quá trình code bạn có sử dụng callback mà không biết đấy (VD như code jQuery, hoặc gọi hàm khi user click vào 1 button trong WinForm, ASP.NET).

Callback có khá nhiều ứng dụng như sau:

- Gọi 1 hàm khi có 1 số sự kiện xảy ra. VD như khi click vào 1 nút ta gọi hàm A, khi tắt cửa sổ ta gọi hàm B, v...v.
- LINQ được xây dựng dựa trên khái niệm callback và lambda expression. Với callback, ta có thể thực hiện các thao tác như: Tìm 1 phần tử trong mảng, lọc phần tử trong mảng, sắp xếp mảng, ... trở nên vô cùng đơn giản và thuận tiện. Các bạn có thể xem thêm tại bài viết [Series C# hay ho: LINQ](#).
-

5 phút quảng cáo cho bộ phim ([series bài viết](#)) sắp tới. Hẹn gặp lại các bạn.



Series C# hay ho: Những điều thú vị trong C# (Phần 1)

Posted on 22/01/2015 by Phạm Huy Hoàng

Như đã giới thiệu, mình là một lập trình viên C#.NET, do đó C# là ngôn ngữ mình tiếp xúc hằng ngày. Lẽ dĩ nhiên, trong quá trình sử dụng + tìm hiểu, mình nhận thấy có C# có những điều nhỏ nhặt, nhưng lại khá hay ho, cho thấy các cụ Microsoft có suy nghĩ tới developer khi tạo ra ngôn ngữ này.

Nội dung bài viết này là những điều “hay ho” mình đã nói. Những điều “hay ho” (mà ít người biết) này giúp cho việc code được nhanh hơn, code dễ đọc dễ hiểu hơn. Trong quá trình so sánh, mình có nhắc tới sự thuận tiện khi developer bằng C# so với “các ngôn ngữ khác”. Ở đây cũng giống như so sánh OMO với “bột giặt thường” vậy, không có ý so sánh C# với Java hay PHP của các bạn đâu nhé, các bạn đừng ném đá tội nghiệp =))).

1. Từ khóa “var” và Anonymous Object

Ngày trước, khi khai báo 1 biến, chúng ta phải khai báo type của biến đó (Ở đây là stream).

```
1 Stream stream = new FileStream("C:\\abc.txt", FileMode.CreateNew);
```

Tuy nhiên, với từ khóa var, chúng ta có thể khai báo biến mà không cần quan tâm đến type. Các chức năng nhắc lệnh vẫn được thực hiện bình thường.

```
1 var stream = new FileStream("C:\\abc.txt", FileMode.CreateNew);
```

Kết hợp với anonymous object, ta có thể khai báo 1 object với các properties định sẵn, không cần phải khai báo class:

```
1 var sampleObj = new { FirstProp = "A", SecondProp = "B" };
2 Console.WriteLine(sampleObj.FirstProp + sampleObj.SecondProp); //AB
```

Anonymous object này có thể được serialize thành XML, JSON. Có thể nói từ khóa “var” là 1 cứu cánh cho những developer “lười” như mình. (Nói nhỏ các bạn nghe, trong Java không có anonymous object cũng như từ khóa “var” này đâu).

2. Tự động tạo Properties

Khi được học về tính bao đóng (Encapsulation) trong trường đại học, chúng ta đều được dạy là không nên access trực tiếp field của object, mà phải thông qua các method như sau.

```
1 public class Student
2 {
3     private string _name;
4     public string Name
5     {
6         get { return _name; }
7         set { _name = value; }
8     }
9 }
```

Với C#, khi ta khai báo 1 auto-property, ngôn ngữ sẽ tự tạo 1 field private, getter và setter cho field đó. Ta còn có thể set private cho getter và setter đó. Code mới như sau:

```
1 public string Name { get; set; }
2
3 public string Name { get; private set; } //Field is read only, private
```

3. using blocks

Ngày xưa ngày xưa, khi chưa có using, mỗi khi muốn đóng 1 stream, connection, ta thường cho vào block try/finally.

```
1 try
2 {
3     stream = new FileStream("C:\\\\abc.txt", FileMode.CreateNew);
4     stream2 = new FileStream("C:\\\\abc2.txt", FileMode.CreateNew);
5     //Do some readding
6 }
7 finally
8 {
9     if (stream != null) stream.Close();
10    if (stream2 != null) stream2.Close();
11 }
```

Sau 1 thời gian, đồng code này sẽ khá rối. Với using, ta không cần phải lo chuyện đóng connection bằng tay, code ngắn và dễ hiểu hơn nhiều

```
1 using(var stream = new FileStream("C:\\\\abc.txt", FileMode.CreateNew))
2 using(var stream2 = new FileStream("C:\\\\abc2.txt", FileMode.CreateNew))
3 {
4     //Do some readding
```

5 }

4. Đo thời gian với class Stopwatch

Đo thời gian 1 method chạy chỉ với 2 dòng code, StartNew và Stop, vô cùng đơn giản.

```
1 var sw = Stopwatch.StartNew();
2 DoSomething();
3 sw.Stop();
4 Console.WriteLine("Time elapsed: " + sw.ElapsedMilliseconds);
```

5. Toán tử 3 ngôi, toán tử null

Thông thường, khi gán giá trị default cho 1 biến null, ta thường set bằng 1 trong 2 cách sau

```
1 if (input == null)
2 {
3     input = "default";
4 }
5
6 // Toán tử 3 ngôi
7 input = input != null ? input : "default";
```

Với C#, ta có toán tử null, giúp code ngắn hơn và dễ đọc hơn

```
1 input = input ?? "default"; //Nếu index là null thì set bằng default
```

Việc implement một toán tử nhỏ nhặt thế này trong ngôn ngữ cho thấy các bạn developer của Microsoft rất biết suy nghĩ cho giới developer chúng ta.

6. Khởi tạo object và collection

Thông thường, khi khởi tạo object và property, ta thường làm như sau

```
1 Student student = new Student();
2 student.Name = "Hoang";
3 student.Age = 10;
```

Với C#, mọi chuyện trở nên đơn giản ngắn gọn hơn

```
1 Student student = new Student { Name = "Hoang", Age = 10};
```

Có thể bạn sẽ hỏi: Tại sao không tạo constructor cho object, cũng vậy thôi mà. Mình xin trả lời: Khi tạo constructor, ta phải khai báo toàn tham số truyền vào. Với cách này, ta có thể truyền vào số lượng tham số ta muốn (Ví dụ object student có 10 fields, ta chỉ muốn set 2 fields).

Tiếp theo là khởi tạo 1 collection, cách thông thường và cách C#. Cách nào ngắn gọn, dễ hiểu hơn các bạn tự thấy nhỉ.

```
1 //Cách cũ
2 List<string> list = newList<string>();
3 list.Add("string 1");
4 list.Add("string 2");
5 list.Add("string 3");
6 //Cách mới
7 List<string> list = newList<string> { "string 1", "string 2",
8 "string 3"};
```

Kết hợp cả class và collection

```
1 List<Student> list = newList<Student>
2 {
3     new Student { Name = "Student 2", Age = 2 },
4     new Student { Name = "Student 3", Age = 4 }
5 };
```

7. Extension method

Có 1 số trường hợp, ta muốn thêm method cho một số class sealed, hoặc class từ các library khác. Với một số ngôn ngữ, điều này là ko thể được, nhưng với C#, chúng ta có thể dùng extension method.

VD ở đây, chúng ta có class Student từ library khác, không thể sửa code. Ta muốn thêm method Print.

```
1 public class Student
2 {
3     public string Name { get; set; }
4     public int Age { get; set; }
5 }
```

Chúng ta tạo 1 extenstion class, class này phải là static class, method cũng phải static, params đầu tiên truyền vào là class cần extention, với từ khóa this.

```
1 public static class StudentExtension
2 {
3     public static void Print(this Student student)
4     {
5         Console.WriteLine(student.ToString());
6     }
7 }
```

```
8  
9 //Sử dụng  
10 var student = new Student();  
11 student.Print();
```

Trong quá trình tìm hiểu, bạn sẽ thấy extension method rất hữu dụng. (*Nói nhỏ nữa là trong Java không có cái extension method này đâu hen*).

8. LINQQQQQQQQQQQQQQQQQQ

Thật ra chỉ có 1 chữ q thôi, nhưng vì đây là 1 trong những điều **cực kì hay ho** của C#, nên mình nhấn mạnh.

Linq khá dễ sử dụng, nhưng để hiểu nó cần biết về Predicate, Func, lambda v...v, mình sẽ dành 1 bài viết để nói rõ hơn. Ở đây mình chỉ giới thiệu sơ, vì sao nó hay.

Ngày xưa, khi chưa có linq, giả sử ta muốn tìm những học sinh có tuổi <20 trong list, ta cần viết code dài như sau:

```
1 var studentsUnder20 = new List<Student>();  
2 foreach (var student in students)  
3 {  
4     if (student.Age < 20) studentsUnder20.Add(student);  
5 }
```

Ngày nay, với linq, ta chỉ cần **đúng 1 dòng code**:

```
1 var studentsUnder20 = students.Where(student => student.Age < 20);
```

Đó chỉ là 1 trong vô vàn những tính năng hay ho của Linq thôi, mình sẽ nói rõ hơn ở bài viết khác. (*Nói nhỏ luôn, trong javascript ta có thể dùng thư việc underscore để có tính năng tương tự. Trong Java 7 trở xuống thì không có Linq, từ Java 8 trở lên mới bắt đầu thêm Linq vào để bắt chước C#*).

Tới đây bài viết cũng đã khá dài. Hẹn gặp lại các bạn ở [phần 2](#) nhé. Mọi ý kiến hay gạch đá đều được tiếp thu nhiệt liệt.

Bài viết được phỏng dịch + lượt từ bản gốc tiếng Anh (Đã xin phép tác giả) : <http://geekswithblogs.net/BlackRabbitCoder/archive/2011/10/24/c.net-little-wonders-the-complete-collection-again.aspx>

Series C# hay ho: Những điều thú vị trong C# (Phần 2)

Posted on 27/01/2015 by Phạm Huy Hoàng

Chào mừng các bạn đã quay lại với phần 2 bài viết **Những điều thú vị trong C#**. Bạn có thể xem lại phần 1 ở đây:

<https://toidicodedao.wordpress.com/2015/01/22/series-c-hay-ho-nhung-dieu-thu-vi-trong-c-phan-1/>

Series C# hay ho là một series dài kì, giới thiệu những điều “hay ho” của C#. Đây chỉ là phần 2 của bài viết thôi, không phải của toàn bộ series đâu nhé.

1. Optional Parameters

Có 1 số bạn sẽ không biết optional parameters là gì (Mình thì biết nó từ C++, tuy nhiên gần đây mới biết là C# cũng có =)). Giả sử không có optional parameter, ta muốn chạy những function Add như sau

```
1     Add(1, 2);
2     Add(1, 2, 3.5);
3     Add(1, 2, 3.5, "Hello");
4     Add(1, 2, true);
```

Ta cần phải viết overload lại function Add những 4 lần.

```
1     public int Add(int num1, int num2)
2     public int Add(int num1, int num2, double num3)
3     public int Add(int num1, int num2, double num3, string num4)
4     public int Add(int num1, int num2, bool bol)
```

Vô cùng phiền phức và mất thời gian phải ko? Với optional parameters, ta có thể viết 1 function nhận vào toàn bộ tham số, và chỉ định 1 số tham số là option, nếu ko đưa vào sẽ sử dụng giá trị mặc định:

```
1     public int Add(int num1, int num2,
2                     double num3 = 0, string num4 = "", bool bol = true)
```

```

3
4     //Câu lệnh Add(1, 2, true);, vì skip num3 và num4, ta sẽ gọi
5     Add(1, 2, bol: false);
6
7     //2 câu lệnh sau sẽ tương đương nhau.
8     //Khi ta không truyền vào num3, num4, chúng sẽ dùng giá trị default
9     Add(1, 2);
10    Add(1, 2, 0, "", true);

```

Nếu các bạn đã học C++ chắc cũng không lại gì với khái niệm này. Như đã thấy, nó giúp chúng ta tiết kiệm được khá nhiều thời gian code. (*Nói nhỏ các bạn nghe, trong Java và javascript không có optional parameters đâu, vì vậy thường thường chúng ta phải overload 1 hàm nhiều lần, rất chi là cực -_-*).

2. Anonymous Type

Như đã nhắc đến ở phần 1, Anonymous Type rất hữu dụng trong nhiều trường hợp. Giả sử chúng ta có 1 class Student với 10 properties.

```

1
2     public class Student
3     {
4         public string Name { get; set; }
5         public int Age { get; set; }
6         public string Name2 { get; set; }
7         public int Age2 { get; set; }
8         public string Field1 { get; set; }
9         public string Field2 { get; set; }
10        public string Field3 { get; set; }
11        public string Field4 { get; set; }
12        public string Field5 { get; set; }
13        public string Field6 { get; set; }
14    }

```

Trường hợp ta chỉ muốn lấy 2 fields Name và Age của Student để serialize ra JSON cho ngắn gọn, ta viết code như sau.

```
1 List<Student> students = new List<Student>();  
2 var stu = students.Select(st => new {Name = st.Name, Age = st.Age});  
3 //Hoặc ngắn gọn hơn  
4 stu = students.Select(st => new {st.Name, st.Age});
```

Anonymous object nhận được sẽ là object với 2 properties: Name và Age. Object này còn được implement sẵn function Equals(), do đó ta có thể so sánh giữa 2 object này. Tuy nhiên, ta cũng cần cẩn thận với một số trường hợp “tưởng bằng mà lại ko bằng”.

```
1 var a = new {Name = "Hoang", Age = 10};  
2 var b = new { Name = "Hoang", Age = 10 };  
3 Console.WriteLine(a == b); // True  
4  
5 //Cẩn thận những trường hợp sau  
6 var c = new { Name = "Hoang", Age = 10 };  
7 var d = new { Name = "Hoang", Age = 10.2 };  
8 var e = new { Age=10, Name = "Hoang", };  
9 Console.WriteLine(c == d); // Không compile, báo lỗi  
10 Console.WriteLine(c == e); // Không compile, báo lỗi
```

3. Tuple

Tuple là một khái niệm khá “lạ”, mình gặp khái niệm này lần đầu ở Python, giờ mới biết trong C# cũng có. Có thể hiểu đơn giản tuple là 1 “cục” tập hợp của nhiều fields, tương tự như class, nhưng define nhanh và dễ hơn. Ví dụ:

```
1 var love = new Tuple<int>(1);  
2 var lie = new Tuple<int, string>(1, "Why");  
3 var lay = new Tuple<int, int, string>(1, 2, "Gay");
```

```
4     Console.WriteLine(lay.Item1 + " " + lay.Item2 + " " + lay.Item3); // 1 2 Gay
```

Anonymous type có 1 khuyết điểm, đó là ko thể dùng anonymous type làm kiểu trả về, hoặc parameter truyền vào cho 1 function.

```
1  public ??? GetStu() //Làm sao trả về
2  {
3      List<Student> students = new List<Student>();
4      var stu = students.Select(st => new { st.Name, st.Age });
5      return stu;
6  }
7
8  DoSomething(stu);
9  public void DoSomething(???) // Làm sao truyền vào
```

Với tuple, vấn đề đó được giải quyết nhanh chóng:

```
1  public List<Tuple<string, int>> GetStu()
2  {
3      List<Student> students = new List<Student>();
4      var stu = students.Select(st => new Tuple<string, int>(st.Name, st.Age)).ToList();
5      return stu;
6  }
7
8  public void DoSomething( List<Tuple<string, int>> input);
9  DoSomething(stu);
```

Có thể các bạn sẽ hỏi: Tại sao ko tạo 1 class mới, dùng anonymous type với tuple làm gì cho mệt? Xin trả lời:

- Có 1 số trường hợp ta cần extract 1 số field, nếu mỗi lần lấy vài fields ta đều tạo class mới thì code sẽ khá dư và vô nghĩa.

- Nếu tạo class mới, ta phải override lại hàm Equals và GetHashCode nếu cần so sánh. Anonymous type và tuples đã tự động implement hàm này cho chúng ta.

4. String và những điều ít người biết

Chắc ai trong số chúng ta cũng đã từng dùng string để vẽ header, footer như thế này nhỉ?

Đoạn code chúng ta đã dùng là

```

1   string header = "";
2   for (int i = 0; i < 80; i++)
3   {
4       header += "";
5   }
6
7   //Cách đơn giản hơn với string, hẳn bạn sẽ rất bất ngờ =))
8   string header = new string('=', 80);

```

Tương tự, khi ta có 1 list các string, và ta muốn cộng toàn bộ các string trong list

```

1   List<string> list = new List<string>{ "1" , "2" , "3" };
2   string result = "";
3   foreach (var str in list)
4   {
5       result += str + ",";
6
7   //Đơn giản hóa bước lần 2
8   string result = string.Join(", ", list);
9

```

```
10     //Bá đạo hơn nữa khi ta join với param không phải string
11     string result = string.Join(", ", "Hello", 1, 2, 3.5); //Hello, 1, 2, 3.5
12
```

6. Khởi tạo Collection (ToArray, ToList, ToDictionary, ...)

Khi sử dụng Linq, sau khi dùng Where, Select, ... kết quả trả về thường là 1 IEnumerable, ta có thể biến chúng thành List hoặc Array với các hàm có sẵn. ToList là hàm được sử dụng nhiều nhất:

```
1     List<Student> students = new List<Student>();
2     IEnumerable<Student> stu = students.Where(st => st.Age > 0);
3
4     //Tạo list
5     List<Student> stuList = stu.ToList();
6
7     //Tạo array
8     Student[] stuArray = stu.ToArray();
9
10    //Tạo 1 dictionary, Key là Name, Value là Age
11    Dictionary<string, int> stuDic = stu.ToDictionary(t => t.Name, t => t.Age);
```

Bài viết tối đây là kết thúc. Vì nội dung bài viết có hạn, một số phần mình chỉ giới thiệu tổng quan, ko đi sâu vào chi tiết. Những phần như **Linq, Anonymous Type, Delegate, ...** sẽ có bài viết cụ thể đầy đủ hơn cho từng phần, mong các bạn đón xem.
Bài viết được phỏng dịch + lượt từ bản gốc tiếng Anh (Đã xin phép tác giả) : <http://geekswithblogs.net/BlackRabbitCoder/archive/2011/10/24/c.net-little-wonders-the-complete-collection-again.aspx>

Series C# hay ho: Callback trong C# – Delegate, Action, Predicate, Func

Posted on 10/02/2015 by Pham Huy Hoàng

1. Nhắc lại về khái niệm callback

Nếu chưa có khái niệm rõ ràng về callback, các bạn nên đọc bài viết về [Callback trong javascript](#) mà mình đã viết.

Trong javascript, để callback, ta chỉ cần truyền 1 function vào như 1 parameter như sau:

```
1 function tangQua(qua) {  
2     return console.log("Đã tăng " + qua);  
3 }  
4  
5 function oNha(vo, tangQua){  
6     var qua = "Quà đã nhận";  
7     tangQua(qua);  
8 }
```

2. Delegate trong C#

Trong javascript, function cũng xem như là 1 object, nên ta có thể truyền function vào như 1 param. Khi ta thử viết lại chương trình trên trong C#, ta phải khai báo kiểu dữ liệu cho parameter truyền vào, vậy kiểu dữ liệu của function “tangqua” là gì?

```
1 public void tangQua(string qua) {  
2     Console.WriteLine("Da tang " + qua);  
3 }  
4  
5 //Kiểu dữ liệu cho params "vo" là Person,  
6 //kiểu dữ liệu cho tangQua là gì???  
7 public void oNha(Person vo, tangQua){  
8     var qua = "Quà đã nhận";  
9     tangQua(qua);  
10 }
```

Ở đây, để truyền function tangQua vào, ta phải sử dụng kiểu dữ liệu Delegate (Con trỏ hàm). Ta khai báo delegate theo cú pháp sau:

```
1 public delegate void TangQuaDelegate(string qua);  
2 //delegate + kiểu trả về (void) + tên delegate + (tham số truyền  
3 vào)
```

```

4
5 public void tangQua(string qua) {
6     Console.WriteLine("Da tang " + qua);
7 }
8 public void onNha(Person vo, TangQuaDelegate tangQua)
9 {
10     var qua = "Quà đã nhận";
11     tangQua(qua);
12 }

```

Trong C#, delegate thường được sử dụng phối hợp với event. Các bạn làm Winform chắc ko lạ lùng với những đoạn code gọi event thế này:

```

1 this.button1.Click += new
2 System.EventHandler(this.button1_Click);
3 //Thật sự thì event handler chỉ là 1 delegate
4 public delegate void EventHandler(object sender, EventArgs e);
5
6 //Function chúng ta truyền vào chính là function với
7 //2 param sender, a, kiểu trả về là void
8 private void button1_Click(object sender, EventArgs e) { }

```

Bài viết này chỉ tập trung về delegate nên các bạn có thể tìm hiểu thêm về delegate và event ở trang khác nhé.

<http://tapchilaptrinh.vn/2012/07/16/delegates-va-events-trong-c/>

3. Action, Predicate, Func trong C#

Ở phần này, mình xin giới thiệu về **Action**, **Predicate**, **Func** (Viết tắt là APF là cho nhanh) trong C#. Như các bạn đã đọc trong [series C# hay ho](#), các lão developer trong Microsoft rất lười. Do đó họ luôn thêm thắt nhiều thứ trong C# để các developer chúng ta cũng lười như họ. APF thật ra là 1 cách **lười hơn** để chúng ta khai báo delegate.

Dưới đây là khái niệm cũng như cách dùng APF :

- **Action**: Action<T in1, T in2, ...>. Action tương đương 1 delegate với kiểu trả về là void, với in1, in2 là các params nhận vào.
- **Predicate**: Predicate<T in1, T in2, ...>. Predicate tương đương 1 delegate với kiểu trả về là bool, với in1, in2 là các params nhận vào

- **Func**: Func<T in1, T in2, ... , T result>. Function tương đương 1 delegate với kiểu trả về do ta khai báo (result), in1, in2 là các params nhận vào. Func bắt buộc phải trả ra giá trị, không thể trả void.

Để dễ hiểu, các bạn hãy tham khảo bảng sau. Đây là bảng so sánh các khai báo bằng delegate, cùng với cách khai báo tương ứng bằng Action, Predicate, Func:

Delegate	Action	Predicate	Func
delegate void VoidDelegate(int input1, bool input2)	Action<int, bool>		
delegate bool BoolDelegate(int input1, bool input2)		Predicate<int, bool>	Func<int, bool, bool>
delegate int intDelegate(bool input2)			Func<bool, int>
delegate void HelloWorldDelegate()	Action		
delegate bool HelloWorldBoolDelegate()		Predicate	Func<bool>

Đoạn code ban đầu có thể được viết lại như sau, ngắn gọn hơn

```

1 public void tangQua(string qua) {
2     Console.WriteLine("Da tang " + qua);
3 }
4
5 public void oNha(Person vo, Action<string> tangQua)
6 {
7     var qua = "Quà đã nhận";
8     tangQua(qua);
9 }
```

Có thể bạn sẽ thắc mắc: Ủa, mình có thấy người khác dùng Action, Predicate, ... mấy đâu, biết làm gì?



Bạn sẽ ngạc nhiên khi biết AFP, kết hợp với lambda expression và vài thứ khác nữa, **đã tạo nên sự “bá đạo”** của **LINQ**, thứ mà Java thèm đỏ mắt mà không có, phải thêm vào ở Java SDK 8. Nhớ đón đọc bài sau “Series C# hay ho: Lambda Expression” để biết thêm chi tiết nhé.

Series C# hay ho: Lambda Expression

Posted on 12/02/2015 by Phạm Huy Hoàng

1. Ôn lại khái niệm Delegate

Các bạn vui lòng đọc lại bài trước để nhớ lại các khái niệm về delegate ở đây: [Series C# hay ho: Callback trong C# – Delegate, Action, Predicate, Func](#)

Như đã nói ở bài trước, delegate là kiểu dữ liệu để trả tới 1 function, do đó khi gán giá trị cho delegate, ta phải gán 1 function vào, như ví dụ dưới đây:

```
1 public delegate void TangQuaDelegate(string qua) ;
2
3 public void tangQua(string qua) {
4     Console.WriteLine("Da tang " + qua);
5 }
6
7 //Khi sử dụng:
8 TangQuaDelegate dlg = tangQua;
9 //Truyền function vào, không phải thực thi function nên ko có dấu
()
```

2. Anonymous function

Như vậy, mỗi lần muốn truyền 1 function vào delegate, ta phải define function đó, khá phiền phức nhỉ? Trong javascript có anonymous method, viết thẳng function mà ko cần define như sau:

```
1 //Dùng anonymous function, tangQuaDlg giờ đã là 1 delegate
2 var tangQuaDlg = function(qua) { alert("qua"); };
3
4 function oNha(vo, tangQua) {
5     var qua = "Quà đã nhận";
6     tangQua(qua);
7 }
8
9 //Sử dụng trong code
10 oNha(vo, tangQuaDlg);
```

May mắn thay, trong C#, ta cũng có thể viết anonymous function theo cách sau (Từ .NET 2.0):

```
1 public delegate void TangQuaDelegate(string qua) ;
2
3 TangQuaDelegate dlg =
4
```

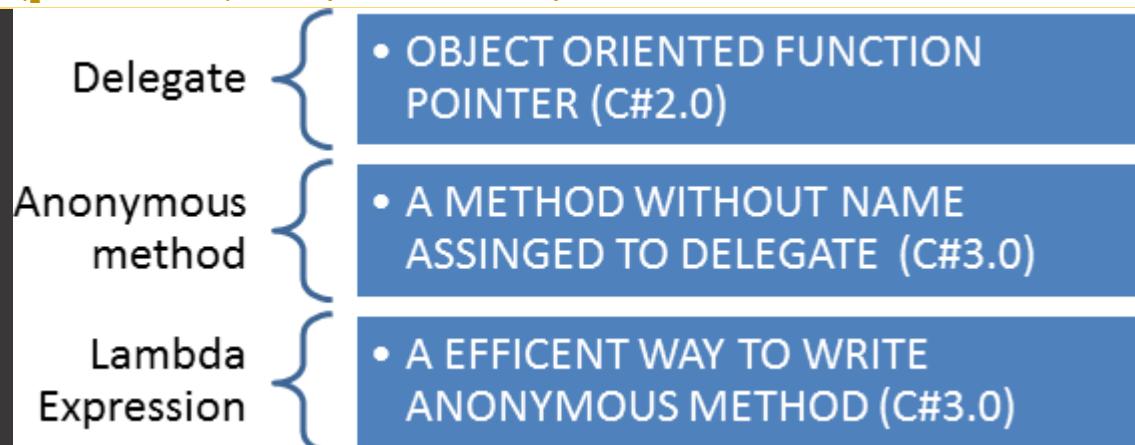
```
delegate(string qua) { Console.WriteLine("Tặng quà" +  
qua); };
```

Ta thấy viết delegate cách này khá là rườm rà phải không? Microsoft cũng thấy vậy, và họ đã bổ sung lambda expression và .NET 3.0. Muốn biết lambda expression là gì, mời xem phần dưới.

3. Lambda expression

Mình muốn nhắc lại 1 chuyện lần thứ n là: “Các lão developer bên Microsoft rất lười, do đó họ thêm vào C#.NET nhiều thứ, làm developer C#.NET lười theo”. Ta có thể hiểu lambda expression là 1 cách viết anonymous function ngắn gọn hơn:

```
1 //Cách cũ  
2 TangQuaDelegate dlg = delegate(string qua)  
3 { Console.WriteLine("Tặng quà" + qua); };  
4  
5 //Dùng lambda expression  
6 TangQuaDelegate lamdaDlg = (qua) => { Console.WriteLine("Tặng  
7 quà: " + qua); }  
8  
9 //Câu lệnh đầy đủ của lambda expression.  
//Đầu "=>" gọi là go-to  
(parameters) => { statement }
```



Dưới đây là một số qui tắc viết lambda expression:

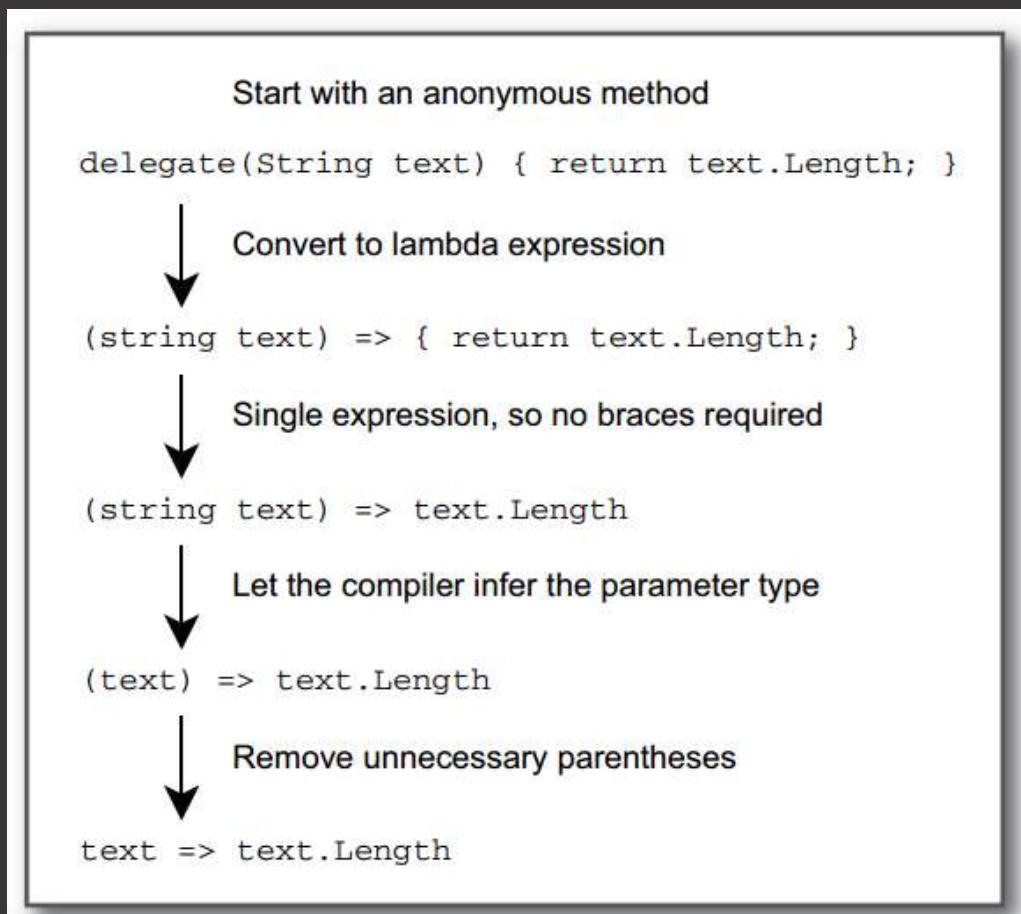
```
1 //1. Có thể bỏ qua kiểu dữ liệu của parameter truyền vào  
2 (string qua) => {Console.WriteLine("Tặng quà: " + qua); }  
3 (qua) => {Console.WriteLine("Tặng quà: " + qua); }  
4  
5 //2. Nếu không có parameter, bỏ dấu () trống  
6 () => {Console.WriteLine("Hello"); }  
7
```

```

8 //3. Nếu chỉ có 1 parameter, có thể bỏ luôn dấu ()
9 (x) => {Console.WriteLine("Hello " + x);}
10 x => {Console.WriteLine("Hello " + x);}
11
12 //4. Nếu có nhiều parameter, ngăn cách bằng dấu phẩy
13 (x, y) => {Console.WriteLine("Hello " + x + y);}
14
15 //5. Nếu anonymous function chỉ có 1 câu lệnh, có thể bỏ dấu {}
16 x => { Console.WriteLine("Hello " + x); }
17 x => Console.WriteLine("Hello " + x)
18
19 //6. Nếu chỉ return 1 giá trị, có thể bỏ chữ return.
20 //4 lambda expression sau tương đương nhau
21 (x) => { return x > 4; }
22 x => { return x > 4; }
23 x => return x > 4
24 x => x > 4

```

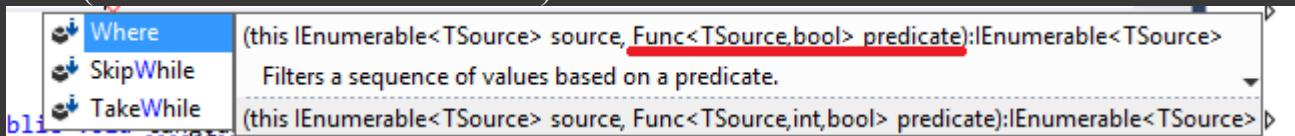
Tới đây, có bạn sẽ ô lèn kinh ngạc: “À, ra vậy”. Đây là cách ta áp dụng các bước trên để rút gọn 1 lambda expression.



Cũng có bạn sẽ thắc mắc: Ô, tưởng lambda expression là gì ghê gớm chứ, hóa ra đơn giản thế thôi à. Vâng, vì nó đơn giản thế nên nhiều bạn sử dụng nó mà **không biết tên gọi** cũng như **chính xác nó là gì**.

4. Lambda Expression và LINQ

Đã có bao giờ bạn xem thử param truyền vào cho 1 hàm của LINQ như Where, First là gì chưa?. Vâng, nó chính là 1 **delegate cho một function có kiểu trả về là bool** (Xem thêm về Func ở bài trước).



Do đó, ta sử dụng lambda expression để truyền 1 anonymous function vào hàm **Where** hoặc **First**. Ví dụ, khi dùng hàm Where của LINQ để tìm những phần tử trong 1 mảng:

```
1 var studentList = new List<Student>();  
2  
3 //Thứ đẹp đẽ ngắn gọn này là lambda expression  
4 var students = studentList.Where(stu => stu.Age > 20);  
5  
6 //Nếu không có nó, ta phải viết cái thứ vừa dài dòng vừa gòm  
7 //ghiếc như sau  
8 var student = studentList.Where(new delegate(Student stu)  
9 { return stu.Age > 20; });  
10  
11 //Hoặc tệ hơn  
public bool FindStudentWithAge(Student stu) { return stu.Age >  
20; }  
var student = studentList.Where(FindStudentWithAge);
```

Chúc mừng các bạn đã biết được thêm 1 điều hay ho mà ít người biết trong C#.NET. Lambda expression cũng như anonymous function là những vấn đề có tính chuyên sâu trong .NET, có thể bạn sẽ bị hỏi nếu muốn apply vào vị trí **senior developer** đấy nhé. Bài viết xin kết thúc ở đây. Hai bài viết tiếp theo sẽ là về Generic và IEnumrable, mong các bạn đón xem.

Series C# hay ho: Generic là cái thứ chi chi

Posted on 05/03/2015 by Phạm Huy Hoàng

Generic là một vị anh hùng thầm lặng trong C#.NET (Dân gian còn gọi là anh hùng núp). Generic 1 trong “5 anh em siêu nhân” cấu thành LINQ (4 người còn lại là:[Extension method](#), [Delegate](#), [Lambda Expression](#) và yield). Anh núp trong 50% những dòng code chúng ta viết, đến nỗi chúng ta dùng 1 cách vô thức, không biết đến sự tồn tại hay tên gọi của anh.

Mình viết bài này nhằm **vinh danh** “anh hùng thầm lặng” Generic, cũng như hướng dẫn các bạn cách **tạo, sử dụng generic class và generic method**.

1. Sự ra đời của Generic

Rất nhiều người trong chúng ta sử dụng generic mà không biết rằng nó là generic. Ví dụ, khi muốn tạo 1 danh sách các học sinh, ta thường viết:

```
1 //Một list chứa các object là Student
2 List<Student> students = new List<Student>();
3 students.Add(new Student()); //Code đúng
4 students.Add(new Car()); //Compile lỗi
5 //Lấy học sinh đầu tiên.
6 //Compiler tự hiểu kết quả là Student
7 Student first = students.First();
```

Có bạn sẽ hỏi: Ô, code này bình thường mà, có thấy cái thằng **anh hùng núp** Generics đâu? Vâng, Generics núp trong 2 dấu ngoặc nhọn đầy bạn <>. Để dễ hiểu, ta hãy quay lại thời .NET 1.0, khi generic chưa xuất hiện:

```
1 //Không có generic
2 //Một list chứa các object
3 List students = new List();
4 students.Add(new Student()); //Compile bình thường
5 students.Add(new Car()); //Compile bình thường
6 //Lấy object đầu tiên, phải ép kiểu sang Student
7 Student first = (Student)students.First();
```

Không có generic, compiler không thể check lỗi lúc compiler. Do đó, ở dòng 2, ta có thể thêm 1 object Car và list gồm các object Student. Khi lấy 1 phần tử ra, ta cũng phải ép kiểu, vì compiler chỉ hiểu nó là 1 object. Vì những lý do đó, generic đã được thêm vào ở .NET 2.0. Tác dụng của generic:

- Giúp tái sử dụng code. Ví dụ: Ta chỉ cần viết class List<T>, T ở đây có thể là bất kì class gì.
- Hỗ trợ compiler bắt lỗi trong quá trình compiler (Hạn chế được tình trạng như dòng 2).
- Không còn phải ép kiểu từ object.
-

2. Sử dụng Generic

Generic được ứng nhiều trong các class List, Dictionary và LINQ,... Khi nào chúng ta nên dùng LINQ? Khi chúng ta cần viết 1 hàm **có thể tái sử dụng cho nhiều kiểu dữ liệu**.

Nghe hơi khó hiểu phải không? Giả sử mình muốn bạn viết 1 hàm swap cho 2 biến, có thể sử dụng cho các kiểu int, double, bool, kể cả class thì bạn sẽ viết thế nào?



Nếu không có generic

```

1 public void swap (int ref a, int ref b);
2 public void swap (double ref a, double ref b);
3 public void swap (bool ref a, bool ref b);
4 public void swap (object ref a, object ref b);
5 .....
6
7 //Nếu muốn swap 2 student, phải viết thêm hàm
8 public void swap (Student ref a, Student ref b);

```

Với sức mạnh của Generic, ta chỉ cần viết 1 và chỉ 1 method duy nhất như sau:

```

1 //Đầu <T> để biết method là generic method
2 public static void Swap<T>(ref T a, ref T b)
3 {
4     T temp = b;
5     b = a;

```

```
6     a = temp;
7 }
```

Sử dụng generic như sau:

```
1 int firstInt = 1;
2 int secondInt = 2;
3 Swap<int>(ref firstInt, ref secondInt);
4
5 double firstDouble = 1.5;
6 double secondDouble = 2.5;
7 //Có thể bỏ qua dấu <> vì compiler tự hiểu kiểu double
8 //Dựa trên tham số ta truyền vào
9 Swap(ref firstDouble, ref secondDouble);
```

Các bạn có thể chạy thử code ở đây: <https://dotnetfiddle.net/7zv1lM>.

Dotnet fiddle cho phép ta viết và chạy code C# online, không cần cài đặt gì nhé.

Một ví dụ nữa, mình vừa gặp trong quá trình code, đó là sử dụng để lấy một giá trị ngẫu nhiên trong list. Ta viết 1 method generic và sử dụng (Method này có thể sử dụng với kiểu int, double, ... cả Student hay object, ...).

```
1 public static T getRandomElement<T>(List<T> list)
2 {
3     Random ran = new Random();
4     int randomIndex = ran.Next(list.Count - 1);
5     return list[randomIndex];
6 }
7
8 //Sử dụng
9 List<int> intList = newList<int> {1,2,3,4,5,6 };
10 List<double> doubleList = newList<double> {0.4,
11 0.6,0.8,4.5,0.2 };
12
13 var randomInt = getRandomElement(intList);
    var randomDouble = getRandomElement(doubleList);
```

Cải tiến 1 chút bằng cách biến method này thành extension method cho class List

```
1 public static class ListExt
2 {
3     public static T randomElement<T>(this List<T> list)
4     {
5         Random ran = new Random();
6         int randomIndex = ran.Next(list.Count - 1);
7         return list[randomIndex];
8     }
9 }
```

```
10
11 //Sử dụng
12 List<int> intList = newList<int> {1,2,3,4,5,6 };
13 List<double> doubleList = newList<double> {0.4,
14 0.6,0.8,4.5,0.2 };
15 //Extension method
16 Console.WriteLine(intList.randomElement());
    Console.WriteLine(doubleList.randomElement());
```

Các bạn có thể test thử đoạn code này tại đây: <https://dotnetfiddle.net/MZJ5HY>

Như đã giới thiệu ở đầu bài viết, Generic là 1 công cụ khá mạnh, có thể tiết kiệm thời gian code, cũng như giảm thiểu khá nhiều bug trong code của bạn. Nếu muốn tìm hiểu thêm, các bạn có thể đọc thêm ở đây: <https://msdn.microsoft.com/en-us/library/512aeb7t.aspx>. Hẹn gặp lại ở bài viết sau.

Series C# hay ho: IEnumerable và yield, tưởng đơn giản mà lầm thứ phải bàn

Posted on 10/03/2015 by Phạm Huy Hoàng

Khái niệm IEnumerable thì chắc cũng có kha khá người biết, khi ta muốn duyệt tất cả các phần tử trong 1 danh sách, ta thường dùng hàm foreach như sau.

```
1   foreach(Student student in students) {}
```

Các kiểu Collection trong C# như List, ArrayList, Dictionary v...v đều implement interface IEnumerable, do đó ta có thể sử dụng foreach để duyệt.

Khái niệm Yield lại được ít người biết tới hơn (Mình thấy rất ít người biết yield là gì, chưa nói đến việc sử dụng). Yield là một keyword thường hay được dùng với IEnumerable. Sử dụng yield sẽ làm code của bạn ngắn gọn, hiệu suất cao hơn rất nhiều. Bài viết này sẽ giải thích cũng như hướng dẫn cách áp dụng từ khóa yield.

1. Nhắc lại về IEnumerable

1 mảng IEnumerable có những thuộc tính sau:

- Là một mảng read-only, chỉ có thể đọc, không thể thêm hay bớt phần tử.
- Chỉ duyệt theo một chiều, từ đầu tới cuối mảng.

Hãy xét trường hợp sau, nếu ta muốn đọc 1 danh sách học sinh từ file, ta thường viết

```
1 public List<Student> ReadStudentsFromFile(string fileName)
2 {
3     string[] lines = File.ReadAllLines(fileName);
4     //Tạo một list trống
5     List<Student> result = new List<Student>();
6
7     foreach(var line in lines)
8     {
9         Student student = ParseTextToStudent(line);
10        result.Add(student); //Thêm student vào list
11    }
12    return result; // Trả list ra
13 }
14
15 var students = ReadStudentsFromFile("students.txt");
```

```
16 foreach(var student in students) {};
```

Đoạn code này không có gì sai. Tuy nhiên ta thấy việc tạo list, thêm phần tử vào list, trả list ra có thể được rút gọn với từ khóa yield như sau

```
1 //Đổi kiểu trả về là IEnumerable
2 public IEnumerable<Student> ReadStudentsFromFile(string fileName)
3 {
4     string[] lines = File.ReadAllLines(fileName);
5     foreach (var line in lines)
6     {
7         Student student = ParseTextToStudent(line);
8         yield return student; //YIELD NÈ
9     }
10 }
11
12 //Dùng như cũ
13 var students = ReadStudentsFromFile("students.txt");
14 foreach(var student in students) {};
```

Bạn sẽ thắc mắc: Ủ, thì rút gọn được 2 dòng code, nhưng mà code có vẻ khó hiểu hơn. Ngày xưa mình cũng nghĩ thế. Ở phần sau, mình sẽ giải thích cơ chế hoạt động của yield, cũng như lý do chúng ta nên dùng yield trong code.

2. Phân biệt return và yield return

Chúng ta đều biết điều cơ bản nhất khi viết 1 method: Từ khóa return sẽ kết thúc method, trả ra kết quả, ko chạy thêm bất kì câu lệnh gì phía sau:

```
1 public int GetNumber() { return 5; }
2 Console.WriteLine(GetNumber());
```

Thế trong trường hợp này, khi chúng ta yield 3 lần thì sao?

```
1 public IEnumerable<int> GetNumber()
2 {
3     yield return 5;
4     yield return 10;
5     yield return 15;
6 }
7 foreach (int i in GetNumber()) Console.WriteLine(i); //5 10 15
```

Sao lạ vậy ta, tại sao ta lấy được cả 3 kết quả? Ta có thể hiểu luồng chạy của chương trình như sau:

- Khi gọi method GetNumber, lấy phần tử đầu tiên, chương trình chạy tới **dòng lệnh số 3**, lấy ra kết quả là 5, in ra console.
- Duyệt tiếp phần tử tiếp theo, chương trình chạy vào **dòng lệnh số 4**, lấy kết quả 10, in ra màn hình.
- Tương tự với phần tử cuối cùng, sau khi in ra, chương trình kết thúc.

Ta hãy quay lại so sánh 2 method đã viết ở đầu chương trình:

```

1 public List<Student> ReadStudentsFromFile(string fileName)
2 {
3     string[] lines = File.ReadAllLines(fileName);
4     List<Student> result = newList<Student>(); //Tạo một list
5     trống
6
7     foreach (var line in lines)
8     {
9         Student student = ParseTextToStudent(line);
10        result.Add(student); //Thêm student vào list
11    }
12    return result; // Trả list ra
13 }
14
15 public IEnumerable<Student> YieldReadStudentsFromFile(string
16 fileName)
17 {
18     string[] lines = File.ReadAllLines(fileName);
19     foreach (var line in lines)
20     {
21         Student student = ParseTextToStudent(line);
22         yield return student;
23     }
24 }
```

- Ở method đầu, ta trả về kết quả sau khi đã chạy hết hàm for, đưa kết quả vào trong 1 list mới, hàm ReadStudentsFromFile kết thúc.
- Ở method thứ 2, kết quả được ngay sau khi parse được student đầu tiên, với mỗi vòng lặp tiếp theo, chương trình sẽ chạy tiếp vào method YieldReadStudentsFromFile, lấy kết quả ra dần dần.

Sau khi đã hiểu bản chất, ta có thể ứng dụng yield vào những trường hợp sau:

- Cần method trả về một danh sách read-only, chỉ đọc, không được thêm bớt xóa sửa.

- Như trường hợp trên, giả sử ta có 50 dòng, hàm **ParseTextToStudent** tốn 1s 1 lần. Với cách cũ, khi gọi hàm **ReadStudentsFromFile**, ta phải đợi 50s. Với hàm **YieldReadStudentsFromFile**, hàm **ParseTextToStudent** chỉ được chạy mỗi khi ta đọc thông tin của học sinh, đó đó tăng performance lên rất nhiều (Nếu ta chỉ lấy 5 học sinh đầu chỉ cần đợi 5s).
- Trong một số trường hợp, danh sách trả về có vô hạn phần tử, hoặc lấy toàn bộ phần tử rất mất thời gian, ta phải sử dụng yield để giải quyết.

Bài viết vừa rồi chỉ hướng dẫn cho bạn khái niệm yield cơ bản. Khi dùng hàm yield, thật ra C# sẽ **compile method** đã viết lại thành 1 **state machine**, implement các method **Next**, **Current**, ... của **IEnumerable**. Bạn nào muốn tìm hiểu thêm có thể đọc thêm ở đây: <http://coding.abel.nu/2011/12/return-ienumerable-with-yield-return/>
Yield là một câu hỏi khá khó nhăn, có thể bạn sẽ bị hỏi khi phỏng vấn vào vị trí **Senior Developer** nhé. Yield cũng là 1 trong “**5 anh em siêu nhân**” tạo nên sự bá đạo của LINQ (4 người còn lại là: [Extension method](#), [Delegate](#), [Lambda expression](#), [Generic](#)). Nếu bạn thường xuyên theo dõi blog, có lẽ bạn đã viết về “5 anh em” này.

Ở bài sau, mình sẽ giới thiệu về LINQ, cũng như lật mặt nạ sự bá đạo nằm sau những method đơn giản như Where, First, ... của nó.

Series C# hay ho: LINQ – Lột mặt nạ sự “bá đạo” của LINQ

Posted on 26/03/2015 by Phạm Huy Hoàng

1. LINQ có thực sự dễ như ta tưởng?

LINQ không phải là 1 khái niệm xa lạ đối với các C#.NET developer, nhất là những bạn hay làm việc với database (LINQ to SQL). Tuy nhiên, đa phần trong chúng ta đều sử dụng LINQ mà không biết rõ nó hoạt động như thế nào. Chắc ai cũng từng viết những dòng code như sau:

```
1 public class Student
2 {
3     public string Name { get; set; }
4     public int Age { get; set; }
5 }
6 var students = newList<Student>();
7 var result = students.Where(stu => stu.Age < 20);
```

Dòng code rất dễ hiểu, lọc ra những học sinh có số tuổi < 20. Tuy nhiên, liệu bạn có trả lời được những câu hỏi sau:

1. Dấu “=>” là dấu gì. Toàn bộ cụm “stu => stu.Age < 20” được gọi là gì?
2. Hàm Where được viết như thế nào, nhận parameter gì vào, trả ra giá trị gì?
3. Tại sao IDE tại biết stu là 1 Student để có thể nhắc lệnh?
4. Có thể viết 1 thứ tương tự như LINQ trong javascript không?

Nếu bạn chỉ trả lời đc 1,2 hoặc không trả lời đc câu hỏi nào, bạn chưa thực sự hiểu LINQ. Đừng lo, ngày xưa mình cũng thế. Bài viết này sẽ giúp bạn giải đáp những câu hỏi trên, cũng như hiểu rõ hơn về LINQ. (**Đáp án ở cuối bài viết**).

2. Ôn lại kiến thức

LINQ được implement bằng cách tổng hợp khá nhiều khái niệm hay ho và phức tạp của ngôn ngữ C#. Nếu bạn nào thường xuyên theo dõi blog, đã đọc những bài viết trước thì chắc là biết rồi. Với các bạn chưa biết, những khái niệm đó bao gồm:

- [Extension method.](#)
- [Callback và Delegate](#) (Func, Predicate).
- [Lamda Expression.](#)

- Generic.
- IEnumerable và yield.

3. Cùng “lột mặt nạ” LINQ

Để sử dụng LINQ, chúng ta phải dùng namespace System.Linq. Thật ra Linq là 1 **danh sách các extension method**, thêm một số method cho interface IEnumerable. Vì các class List, DbSet, ... đều implement interface này, do đó chúng đều có thể gọi Linq method.

Đây là signature của 3 Linq method cơ bản: **Where**, **Select** và **First**.

```

1 public static class Enumerable
2 {
3     public static IEnumerable<TSource> Where<TSource>(this
4     IEnumerable<TSource> source, Func<TSource, bool> predicate);
5     public static IEnumerable<TResult> Select<TSource,
6     TResult>(this IEnumerable<TSource> source, Func<TSource, TResult>
7     selector);
8     public static TSource First<TSource>(this IEnumerable<TSource>
9     source, Func<TSource, bool> predicate);
10 }
```

Có bạn sẽ ngạc nhiên: Cái đồng xấu xí gớm ghiếc khó hiểu này chính là Linq xinh đẹp của mình sao? Vâng, ngày xưa sau khi xem bạn gái tẩy trang xong mình cũng có cảm giác như các bạn vậy =)))

Trong phạm vi bài viết, mình chỉ giải thích method Where, các bạn có thể dựa vào đó để tự tìm hiểu thêm. Ta thấy method nhận vào 1 object có kiểu **Func<TSource, bool>** (link), có a thấy quen ko? Đây là 1 delegate (con trỏ hàm), trả tới 1 method có params là TSource, kiểu trả về là bool. TSource ở đây là kiểu dữ liệu generic, do đó ta có thể khai báo kiểu gì cũng được.

Nếu bạn chưa rành lầm về generic, hãy xem kiểu TSource như kiểu Student, signature của hàm Where sẽ thành:

```

1 public static IEnumerable<Student> Where<Student>(this
2     IEnumerable<Student> source, Func<Student, bool> predicate);
```

Mình sẽ viết lại method Linq ở đầu bài, từ phức tạp đến đơn giản. Mình không dùng var để các bạn hiểu rõ kiểu dữ liệu đưa vào và trả về:

```

1 //Gán method vào delegate
2
```

```

3 public bool FindStudent(Student stu) { return stu.Age < 20; }
4
5
6 //Sử dụng lambda expression sẽ ngắn hơn
7 //Xem lại bài viết về lamda expression
8 Func<Student, bool> func1 = FindStudent;
9 func1 = stu => stu.Age < 20;
10
11 List<Student> students = new List<Student>();
12 IEnumerable<Student> result = students.Where<Student>(func1);
13
14 //Với generic trong C#, ta không cần nói rõ kiểu dữ liệu.
15 //Dựa vào kiểu dữ liệu của Func<Student>, truyền vào
16 bool&gt; truyền vào
17 //C# tự hiểu ta dùng hàm Where với kiểu dữ liệu Student

//Ta bỏ kiểu Student, thay func bằng lambda
var result = students.Where(stu => stu.Age < 20);

```

4. Tự viết LINQ

Đến đây, chúng ta có thể tự viết 1 Linq method . Mọi chuyện rất đơn giản, bạn chỉ cần viết 1 extension method cho I Enumerable, với signature tương tự như Linq. Đây là method Linq do mình viết

```

1 public static class MyLinq
2 {
3     //Extension method
4     public static IEnumerable<TSource> MyWhere<TSource>(this
5     IEnumerable<TSource> source, Func<TSource, bool> predicate)
6     {
7         //Loop toàn bộ cái item trong danh sách truyền vào
8         foreach (var item in source)
9         {
10             // Callback lại hàm đã truyền vào, hàm này trả giá trị
11             boolean
12                 // Nếu hàm callback return true, đưa item đó vào
13             IEnumerable kết quả
14                 // Xem lại vài IEnumerable và yield
15                 if (predicate(item)) yield return item;
16             }
17         }
18     }

```

Method này khá đơn giản, vì mình đã bỏ qua các bước check null v...v, tuy nhiên nó cho kết quả tương tự như method Where trong Linq.

```
1     var result = students.MyWhere(stu => stu.Age < 20);
```

Tới đây, mình sẽ trả lời 4 câu hỏi đã nêu ra ở đầu bài.

1. **Dấu “=>” là dấu gì. Toàn bộ cụm “stu => stu.Age < 20” được gọi là gì?** Dấu “=>” là dấu “go-to”, toàn bộ cụm đi kèm là 1 lambda expression. Bản chất của lambda expression là 1 anonymous method.
2. **Hàm Where được viết như thế nào, nhận parameter gì vào, trả ra giá trị gì?** Hàm Where là 1 extension method của class IEnumerable, nhận vào 1 delegate với kiểu trả về là bool, trả ra 1 IEnumerable đã filter.
3. **Tại sao IDE tại biết stu là 1 Student để có thể nhắc lệnh?** Ở đây ta sử dụng lambda expression, stu là 1 param của hàm. Do Linq sử dụng generic, list students là 1 list với kiểu dữ liệu Student. do đó C# tự hiểu stu là 1 param với kiểu dữ liệu Student.
4. **Có thể viết 1 thứ tương tự như LINQ trong javascript không?** Có thể. Bạn có thể tìm hiểu thêm về underscore và lodash, 2 thư viện hỗ trợ linq cho javascript.

Bài viết kết thúc ở đây. Bạn nào muốn tìm hiểu chuyên sâu hơn có thể tìm đọc [C# in Depth](#). 70% kiến thức mình có được về LINQ và cái khái niệm trên là nhờ đọc cuốn này . Vì LINQ là 1 chủ đề khá phức tạp, cần giải thích nhiều, nếu bạn nào còn thắc mắc có thể để lại comment hoặc gửi mail cho mình nhé.

Series C# hay ho – Tổng quan về Windows Services (WS)

Posted on 15/12/2015 by Phạm Huy Hoàng

Đây là một bài viết khá hay của bạn **Phạm Hồng Sang**, bạn cùng lớp ở FPT và đồng nghiệp [ASWIG](#) với mình. Bài viết sẽ cho bạn cái nhìn tổng quan về Windows Service, cũng như cách viết một Window Services để xử lý các tác vụ chạy ngầm.

Windows Services là gì?

- Windows Service có thể hiểu nôm na như là 1 ứng dụng chạy nền trong một khoảng thời gian dài từ khi bạn bật máy tính cho đến khi tắt nó đi.

- Nó có thể tự động chạy khi máy tính được boots lên, có thể restart hay pause mà không cần một sự tác động nào của người dùng tới các công cụ liên quan tới UI.
- Có thể cài đặt dễ dàng nhờ công cụ hỗ trợ có sẵn của Visual Studio exe thông qua Command Line. Bạn chỉ cần trỏ đúng thư mục có sẵn của file exe và execute nó. Thế là bạn đã cài đặt xong Windows Service.

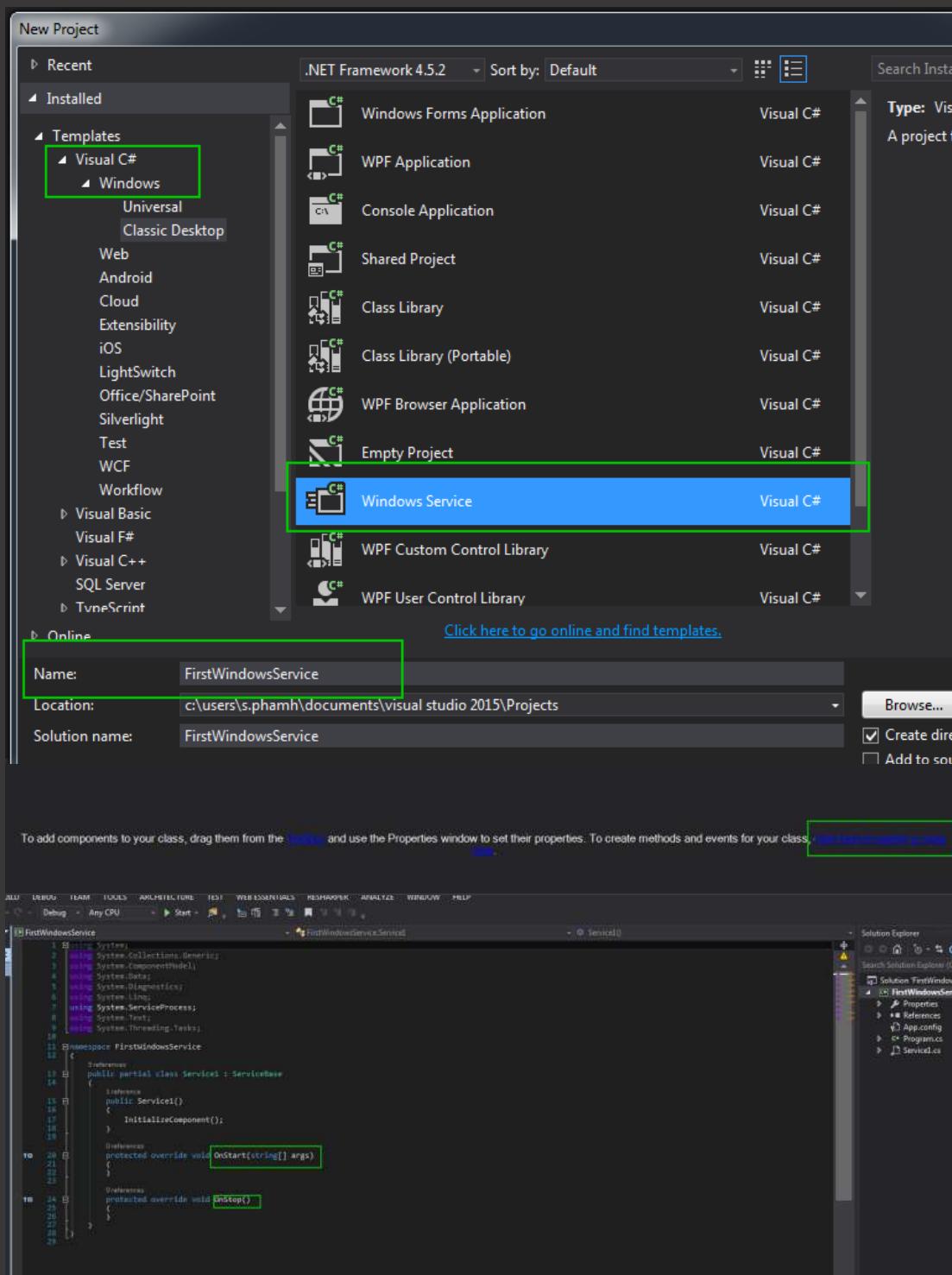
Tại sao phải sử dụng WS

- Một trong những yêu cầu quan trọng của khách hàng là: có một ứng dụng chạy ngầm mỗi ngày để tổng hợp data dùng để xuất ra report, đồng bộ data từ các server, xem hôm nay đã có bao nhiêu sản phẩm được bán, hoặc là gửi mail cho các người dùng khi tài khoản của họ đã hết thời gian sử dụng, v...v... :D
- Xây dựng 1 WS có thể được xem là 1 giải pháp tối ưu cho những tình huống ở trên mà không cần người dùng thao tác gì tới WS cả, mọi thứ đều được chạy ngầm trên máy tính/server của họ.

Viết một Window Services đơn giản

Với những vấn đề nêu trên, bài viết này mình sẽ hướng dẫn các bạn xây dựng 1 WS từ cơ bản tới nâng cao từng bước từng bước 1 (đây là những gì mình học được từ dự án thực tế nên muốn share với các bạn :D).

Đầu tiên chúng ta sẽ tạo 1 Project Windows Service như sau.



*Note: Vì mình xài VS 2015 nên trong Windows sẽ có Universal + Classic Desktop. Còn trong VS 2012-2013 thì ko có nhé :D.

Sau khi tạo xong project, tiếp theo chúng ta sẽ tạo 1 class *Utilities* để ghi vào file Log những sự kiện xảy ra của WS như sau.


```
29      }
30
31
32
33
```

Tiếp theo chúng ta sẽ trả lại class *Service1.cs* để bắt đầu implement phần chính của Service. Trong bài mở đầu này, mình sẽ làm những bước cơ bản trước để các bạn hình dung flow của Windows Service trước nhé.

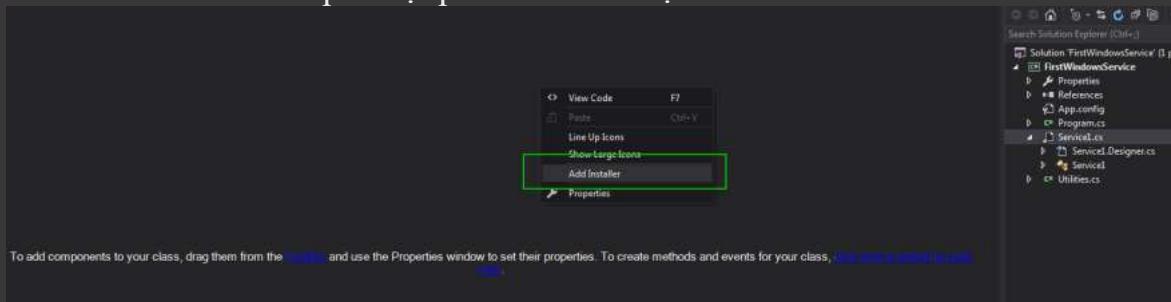
```
1  public class Service1
2  {
3      private Timer timer = null;
4
5      public Service1()
6      {
7          InitializeComponent();
8
9      }
10
11     protected override void OnStart(string[] args)
12     {
13         // Tạo 1 timer từ libary System.Timers
14         timer = new Timer();
15         // Execute mỗi 60s
16         timer.Interval = 60000;
17         // Những gì xảy ra khi timer đó dc tick
18         timer.Elapsed += timer_Tick;
19         // Enable timer
20         timer.Enabled = true;
21         // Ghi vào log file khi services dc start lần đầu tiên
22         Utilities.LogError("Test for 1st run WindowsService");
23     }
24
25     private void timer_Tick(object sender, ElapsedEventArgs args)
```

```

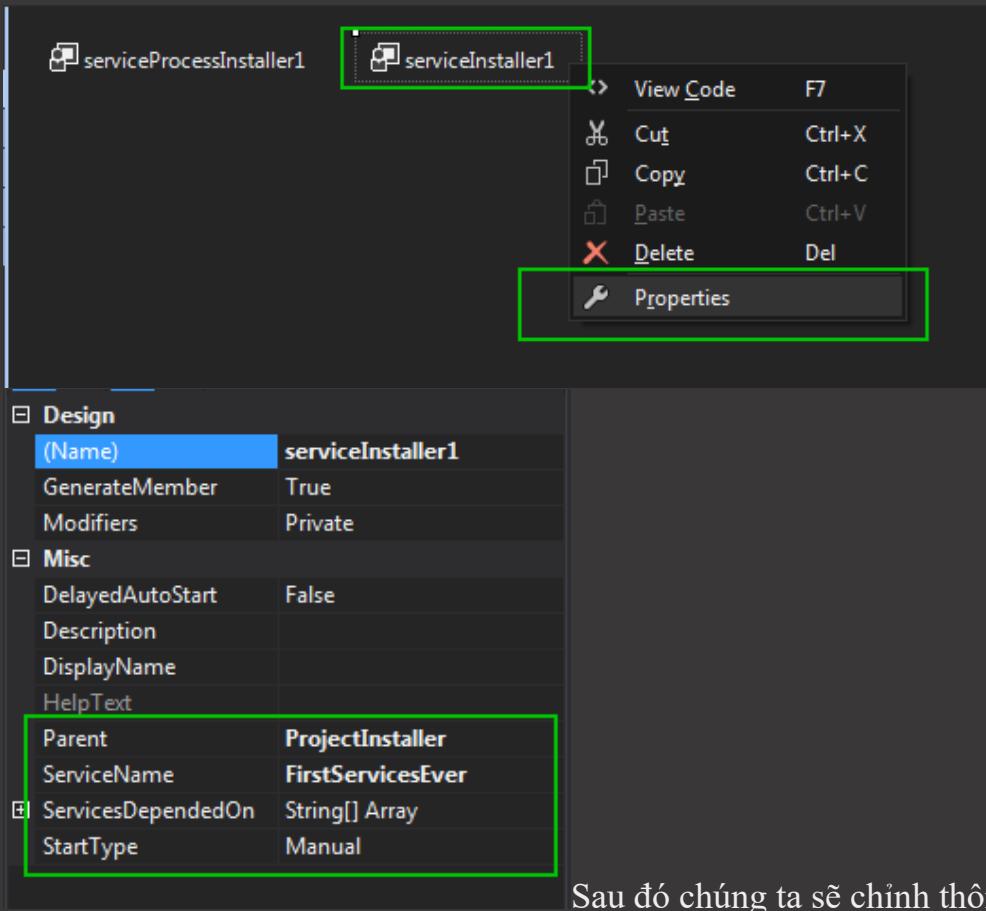
22         {
23             // Xử lý một vài logic ở đây
24             Utilities.Write.LogError("Timer has ticked for doing something!!!");
25         }
26
27         protected override void OnStop()
28     {
29         // Ghi log lại khi Services đã được stop
30         timer.Enabled = true;
31         Utilities.Write.LogError("1st WindowsService has been stop");
32     }
33
34
35

```

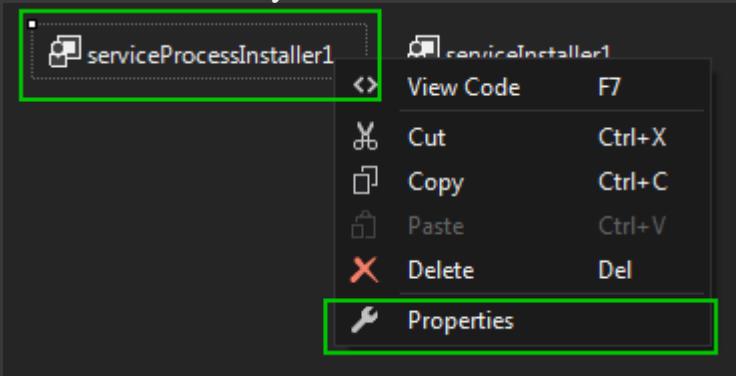
Tới đây thì khá đơn giản để hiểu phải không nào, chúng ta sẽ gọi hàm *timer_Tick* mỗi 60s để thực hiện 1 business gì đó. Tiếp theo chúng ta trở về UI design của *Service1.cs* và nhấp chuột phải lên đó để tạo 1 Installer cho WS của mình.



Sau khi tạo xong 1 Installer thì chúng ta sẽ vào properties để cấu hình những thứ cần thiết ví dụ như *ServicesName* và *StartType*. Trong ví dụ này mình sẽ chọn mode là Manual để demo, các bạn có thể dùng automatically hay disabled.



Sau đó chúng ta sẽ chỉnh thông số cho Process Installer như sau. Vì để demo nên mình sẽ sử dụng máy local để thực thi nên sẽ chọn Account là LocalSystem.

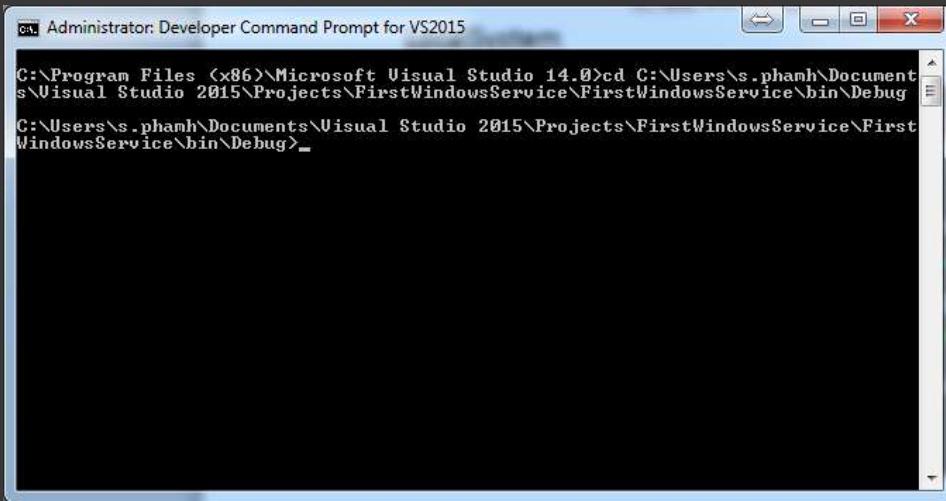


Design	
(Name)	serviceProcessInstaller1
GenerateMember	True
Modifiers	Private
MISC	
Account	LocalSystem
Help Text	
Parent	ProjectInstaller

Các bạn đã viết xong 1 Windows Service rồi đây. Build project và đọc phần sau để biết cách cài đặt Windows Service nhé các bạn.

Cài đặt Windows Service

Các bạn mở Command line của Visual Studio 2015 như sau : Start -> All Programs -> Visual Studio 2015 -> Visual Studio Tools -> Developer Command Prompt for VS2015. Tới thư mục debug của Project Windows Service.



```

Administrator: Developer Command Prompt for VS2015
C:\Program Files (x86)\Microsoft Visual Studio 14.0>cd C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug
C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug>_

```

Tại đây chúng ta sẽ dùng lệnh Install, phía sau Install là tên file exe chứa Service nhé:

InstallUtil "FirstWindowsService.exe"

```
c:\ Administrator: Developer Command Prompt for VS2015
C:\Program Files (x86)\Microsoft Visual Studio 14.0>cd C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug
C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug>InstallUtil "FirstWindowsService.exe"
```

```
c:\ Administrator: Developer Command Prompt for VS2015
C:\Program Files (x86)\Microsoft Visual Studio 14.0>cd C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug
C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug>InstallUtil "FirstWindowsService.exe"
Microsoft (R) .NET Framework Installation utility Version 4.6.81.0
Copyright (C) Microsoft Corporation. All rights reserved.

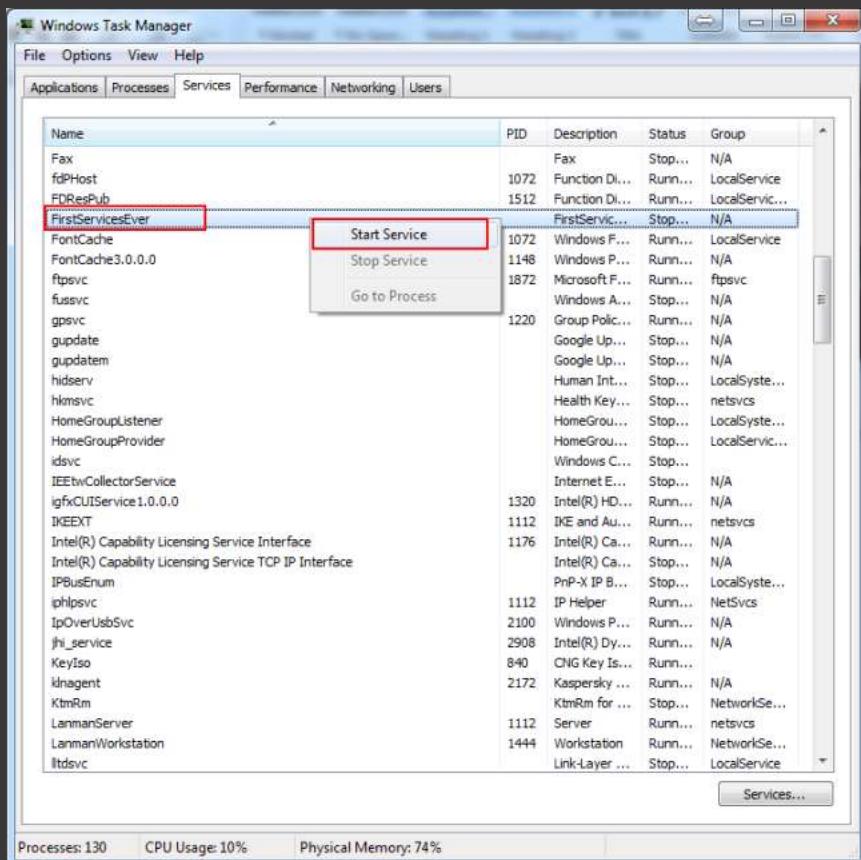
Running a transacted installation.

Beginning the Install phase of the installation.
See the contents of the log file for the C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug\FirstWindowsService.exe assembly's progress.
The file is located at C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug\FirstWindowsService.InstallLog.
Installing assembly 'C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug\FirstWindowsService.exe'.
Affected parameters are:
    logtoconsole =
        logfile = C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug\FirstWindowsService.InstallLog
        assemblypath = C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug\FirstWindowsService.exe
Installing service FirstServicesEver...
Service FirstServicesEver has been successfully installed.
Creating EventLog source FirstServicesEver in log Application...

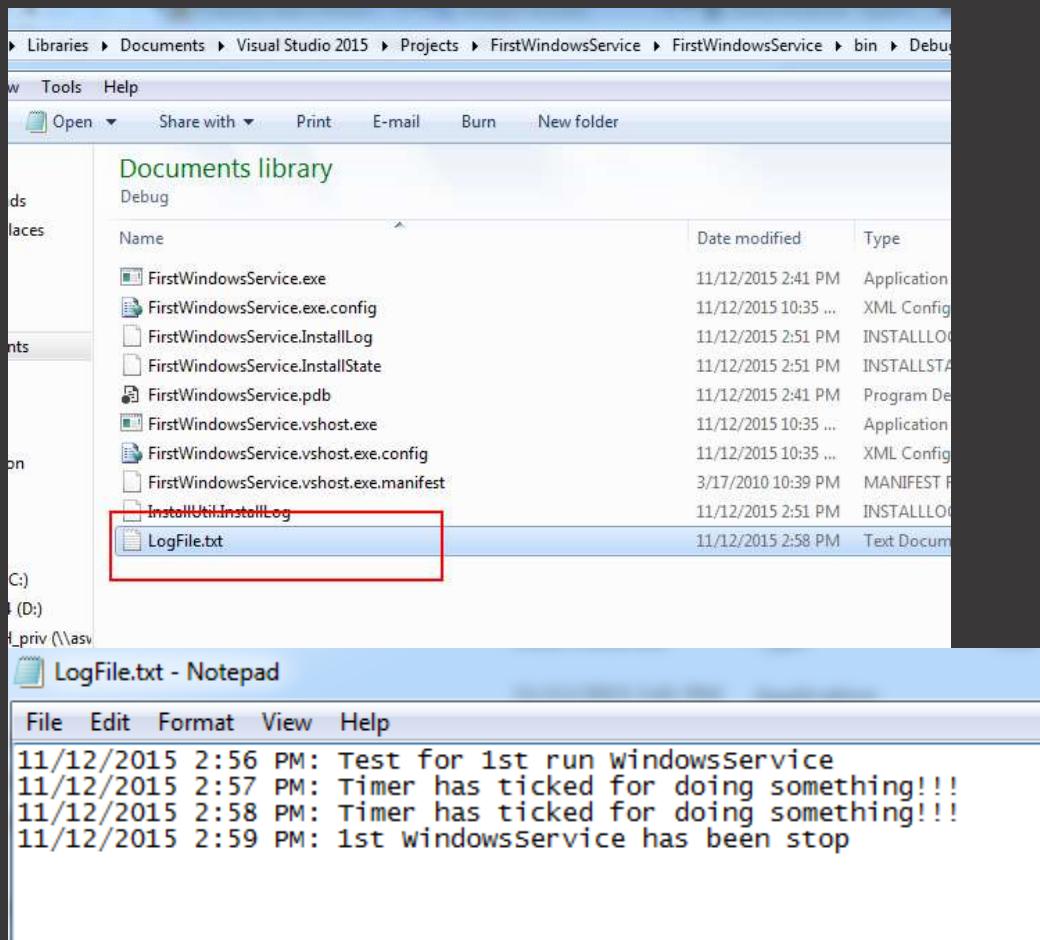
The Install phase completed successfully, and the Commit phase is beginning.
See the contents of the log file for the C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug\FirstWindowsService.exe assembly's progress.
The file is located at C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug\FirstWindowsService.InstallLog.
Committing assembly 'C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug\FirstWindowsService.exe'.
Affected parameters are:
    logtoconsole =
        logfile = C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug\FirstWindowsService.InstallLog
        assemblypath = C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug\FirstWindowsService.exe
The Commit phase completed successfully.
The transacted install has completed.

C:\Users\s.phamh\Documents\Visual Studio 2015\Projects\FirstWindowsService\FirstWindowsService\bin\Debug>
```

Nếu thấy kết quả như trên hình là các bạn đã install thành công rồi đó. Bước cuối cùng là Start Services đó lên và xem file log nhé.



File log đã được generate ra trong chính folder chứa service đó. Các event đều đã được log lại trong file .



Tới đây thì chắc là các bạn đã hiểu cách build một WS như thế nào rồi nhỉ. Để dễ
tham khảo Các bạn có thể lấy code của toàn bộ project ở

đây: <https://github.com/ToiDiCodeDaoSampleCode/WindowService-sample>.

Trong phần sau mình sẽ hướng dẫn các bạn sử dụng thư viện Quartz để hẹn giờ cho 1
WS và cũng là những gì mình đã làm mà đúc kết được :D. Hẹn gặp lại các bạn nhé.

Series C# hay ho: Giới thiệu Humanizer – Một thư viện khá thú vị của C#

Posted on 11/08/2015 by Phạm Huy Hoàng

Kì này, series [C# hay ho](#) sẽ giới thiệu với bạn một thư viện khá “hay ho”, đúng như tên gọi của series. Thư viện này có tên là [Humanizer](#). Nó chỉ có một chức năng duy nhất: Chuyển string, ngày tháng, ... thành **chữ mà con người đọc được** (Đúng như tên gọi Humanizer tức là “người hóa”). Nghe đơn giản vậy thôi, nhưng bạn sẽ ngạc nhiên trước những chức năng của nó.

Bài viết chỉ mang tính chất showcase, giới thiệu nên sẽ không có code nhiều. Nếu tò mò, các bạn có thể tạo 1 project mới, dùng nuger để cài đặt Humanizer và code thử nhé. Một số tính năng nổi bật của Humanizer

1. Chuyển tên hàm, tên biến thành chuỗi có nghĩa

Trong lập trình, ta hay đặt tên thường, tên hàm, tên biến theo naming convention. Humanizer cho phép biến những tên này thành một chuỗi có nghĩa

```
1 "PascalCaseInputStringIsTurnedIntoSentence".Humanize();  
2 //Pascal case input string is turned into sentence  
3  
4 "Underscored_input_string_is_turned_into_sentence".Humanize();  
5 //Underscored input string is turned into sentence  
6  
7 "Underscored_input_String_is_turned_INTO_sentence".Humanize();  
8 //Underscored input String is turned INTO sentence
```

Ta cũng có thể chuyển đổi ngược lại bằng các hàm Pascalize, Camelize, Underscore

```
1 "some title".Pascalize(); //SomeTitle  
2 "some title".Camelize(); //someTitle  
3 "some title".Underscore(); //some_title
```

Có thể transform một string thành thành: chữ hoa, chữ thường, viết hoa đầu dòng, đầu mỗi chữ

```
1 "Sentence casing".Transform(To.LowerCase); //sentence casing  
2 "Sentence casing".Transform(To.SentenceCase); //Sentence casing  
3 "Sentence casing".Transform(To.TitleCase); //Sentence Casing  
4 "Sentence casing".Transform(To.UpperCase); //SENTENCE CASING
```

2. Thu gọn chuỗi

Để hiển thị thông tin dài, ta thường viết hàm cắt chuỗi, sau đó kết thúc bằng dấu ...
Humanizer có sẵn hàm hỗ trợ việc này. Ta còn có thể cắt theo số kí tự hoặc số từ

```
1 "Long text to truncate".Truncate(10, Truncator.FixedLength); //Cắt 10 kí tự
2 //Long text...
3 "Long text to truncate".Truncate(2, Truncator.FixedNumberOfWords); //Cắt 2 từ
4 //Long text...
```

3. Chuyển enum thành chuỗi có nghĩa

Với 1 số enum dài, ta thường dùng Annotation để chuyển nó thành chuỗi có nghĩa.
Humanizer xử lý chuyện này rất dễ dàng.

```
1 public enum EnumUnderTest
2 {
3     [Description(Custom description)]
4     MemberWithDescriptionAttribute,
5     MemberWithoutDescriptionAttribute,
6 }
7 // Nếu có DescriptionAttribute thì đọc từ Attribute
8 EnumUnderTest.MemberWithDescriptionAttribute.Humanize(); //Custom
9 description
10 // Nếu không có thì tự chuyển tên enum sang chuỗi
11 EnumUnderTest.MemberWithoutDescriptionAttribute.Humanize(); //Member without
description attribute
```

4. Xử lý DateTime

Bạn muốn làm chức năng tính thời gian post của comment như facebook (1 giờ trước,
2 giờ trước, v...v). Humanizer có chức năng này, hỗ trợ cả tiếng Anh lẫn Việt :o

```
1 Thread.CurrentThread.CurrentCulture = new CultureInfo("vi-VN"); //Set ngôn
2 ngữ Việt
3
4 DateTime.UtcNow.AddHours(-30).Humanize(); // "hôm qua"
5 DateTime.UtcNow.AddHours(-60).Humanize(); //"cách đây 2 ngày
6
7 DateTime.UtcNow.AddHours(2).Humanize(); //2 giờ nữa
8 DateTime.UtcNow.AddDays(1).Humanize(); //Ngày mai
```

5. Xử lý TimeSpan

Chức năng xử lý TimeSpan cũng tương tự, cả tiếng anh lẫn tiếng Việt đều chạy rất tốt.

```
1 TimeSpan.FromMilliseconds(2).Humanize(); //2 milliseconds
2 TimeSpan.FromDays(1).Humanize(); //1 day
3 TimeSpan.FromDays(16).Humanize(); //2 weeks
4
5 // Mặc định, hàm Humanize() sẽ đưa ra đơn vị lớn nhất (năm >; tháng -
6 &gt;; tuần, ...).
```

```
7 // Ta truyền thêm tham số để hiển thị các đơn vị nhỏ hơn
8 TimeSpan.FromMilliseconds(1299630020).Humanize(); //2 weeks
9 TimeSpan.FromMilliseconds(1299630020).Humanize(3); //2 weeks, 1 day, 1 hour
10 TimeSpan.FromMilliseconds(1299630020).Humanize(4); //2 weeks, 1 day, 1 hour,
11 30 seconds
```

6. Chuyển số thành chữ (Chỉ tiếng Anh)

Nhìn code là hiểu hen, tiếc là không có tiếng Việt :(

```
1 1.ToWords(); //one
2 10.ToWords(); //ten
3 11.ToWords(); //eleven
4 122.ToWords(); //one hundred and twenty-two
5 3501.ToWords(); //three thousand five hundred and one
```

7. Xử lý file size (Khá hay)

Humanizer cho phép xử lý các số liệu liên quan tới filesize (KB, MB, GB, ...) một cách dễ dàng

```
1 var fileSize = (10).Kilobytes();
2
3 fileSize.Bits      =&gt; 81920
4 fileSize.Bytes     =&gt; 10240
5 fileSize.Kilobytes =&gt; 10
6
7 //Ta có thể cộng trừ file size
8 var total = (10).Gigabytes() + (512).Megabytes() - (2.5).Gigabytes();
9
10 //Hiển thị dưới dạng text
11 7.Bits().ToString();           // 7 b
12 8.Bits().ToString();           // 1 B
13 (.5).Kilobytes().Humanize();   // 512 B
14 (1000).Kilobytes().ToString(); // 1000 KB
15 (1024).Kilobytes().Humanize(); // 1 MB
16
17 //Parse ngược từ text ra (Không phân biệt hoa thường)
18 ByteSize.Parse("1.55 mB");
19 ByteSize.Parse("1.55 mb");
20 ByteSize.Parse("1.55 GB");
21 ByteSize.Parse("1.55 gB");
22 ByteSize.Parse("1.55 gb");
23 ByteSize.Parse("1.55 TB");
```

Trong phạm vi bài viết, mình chỉ giới thiệu được một số tính năng nổi bật của Humanizer. Các bạn có thể tìm hiểu thêm ở đây: <https://github.com/MehdiK/Humanizer>. Hẹn gặp lại.

Series C# hay ho: C# 6 có gì hay ho nào

Posted on 22/10/2015 by Phạm Huy Hoàng

Gần đây, mình tập trung viết một số bài về [định hướng nghề nghiệp](#) và [phát triển bản thân](#) mà ít viết về công nghệ. Vì vậy hôm nay mình quyết định tiếp tục viết series [C# hay ho](#). Ở bài này, mình sẽ giới thiệu những cải tiến được Microsoft tích hợp vào C# 6. Bạn nào muốn biết thêm về sự phát triển của C# từ bản 1.0 lên 5.0 hãy đọc bài cũ của mình ở [đây](#) nhé.

Ở phiên bản này, C# không có nhiều cải tiến to lớn như [Linq](#), hay `async/await`. Những thay đổi của bản 6 giúp việc viết code dễ hơn, code ngắn gọn và đẹp hơn.

1. Hàm `nameof()`

Đôi khi, ta cần lấy tên của một param dưới dạng string, trước đây ta phải hardcoded như sau.

```
1 public void Delete(int id)
2 {
3     //Call database to delete
4     // "id" là hardcoded tên param id
5     _log.Info("id " + id + "is deleted");
6 }
```

Khi ta refactor code, đổi tên param, ta phải sửa lại phần đã hardcoded.

```
1 public void Delete(int studentId)
2 {
3     //Call database to delete
4     _log.Info("studentId " + id + "is deleted");
5 }
```

Giờ đây với hàm `nameof`, ta chỉ cần refactor và đổi tên param là được

```
1 public void Delete(int studentId)
2 {
3     //Call database to delete
4     _log.Info(nameof(studentId) + id + "is deleted");
5 }
```

2. Khai báo giá trị mặc định cho property

Trước đây, để khai báo giá trị mặc định cho 1 property, ta phải khai báo trong constructor.

```
1 public class Student {
2
3     public int Age { get; set; }
```

```
4 //Khai báo giá trị mặc định trong instructor
5 public Student()
6 {
7     Age = 18;
8 }
9 }
10 }
```

Giờ đây ta đã có thể khai báo một cách dễ dàng

```
1 public class Student {
2     //Quá nhanh quá nguy hiểm
3     public int Age {get;set;} = 10;
4 }
```

3. Khởi tạo danh sách với index

Một số kiểu collection như Map, Dictionary nhận dữ liệu vào dưới dạng key, value.

Với C# 5, bạn sẽ phải gọi hàm khởi tạo như sau:

```
1 //Khó đọc
2 var idsAndNames = new Dictionary<int,string>{
3     {1, "Hoang"},
4     {2, "Minh"},
5     {3, "Hung"},
6 };
```

C#6 ta cho phép ta khởi tạo list với Index, với cú pháp dễ đọc hơn nhiều.

```
1 var idsAndNames = new Dictionary<int,string>{
2     [1] = "Hoang",
3     [2] = "Minh",
4     [3] = "Hung"
5 };
```

4. Cú pháp using static

Khi viết ứng dụng **Console**, chắc đôi lúc bạn cũng bức mình vì phải
viết **Console.?** lặp đi lặp lại nhiều lần nhỉ?

```
1 Console.ForegroundColor = ConsoleColor.Red;
2
3 foreach(int num in Enumerable.Range(1,10)){
4     Console.WriteLine("Square of " + num + " is: " + Math.Pow(num,2));
5 }
```

Với phiên bản mới, ta có thể loại bỏ việc lặp đi lặp lại bằng cách khai báo **Console**,
Math, ... ở mục **using**, sử dụng **using static**

```
1 using static System.Console;
```

```

2  using static System.Linq.Enumerable;
3  using static System.Math
4
5  //Code ngắn hơn nhiều
6  ForegroundColor = ConsoleColor.Red;
7  foreach(int num in Range(1,10)){
8      Console.WriteLine("Square of " + num + " is: " + Pow(num,2));
9  }

```

Tuy nhiên, nếu dùng nhiều quá, code bạn sẽ khó đọc, dễ nhầm lẫn và conflict giữa các using với nhau. Vì vậy nhớ đừng lạm dụng nhé.

5. Dùng lambda expression để khai báo property và method

Với một số function hoặc property của method, ta phải viết một method có return đầy đủ.

```

1  public class Student {
2
3      public string Name {get; set;}
4      public int Age {get; set;}
5
6      public bool IsOld {
7          get { return Age >=20; }
8      }
9      public string ToString(){
10         return "Name: ." + Name +"Age: " + Age;
11     }
12 }
13

```

Giờ đây, ta có thể khai báo method đó bằng lambda expression. Lưu ý là ta chỉ nên áp dụng cách này cho các properties/method đơn giản. Những method dài thì hãy viết kiểu cũ.

```

1  public class Student {
2
3      public string Name {get; set;}
4      public int Age {get;set;}
5
6      //Sử dụng lambda expression
7      public bool IsOld => Age >=20;
8      public string ToString() => "Name: ." + Name +"Age: " + Age;
9
10 }

```

Bạn nào xem không hiểu lắm thì đọc lại bài viết về lambda expression của mình nhé.

6. Format string ngắn gọn

Cá nhân mình thấy đây là cải tiến hay thứ nhì của bản 6. Ngày xưa, khi cần tạo một string, ta thường phải cộng chuỗi, hoặc sử dụng **string.Format**.

```
1 //Class student
2 public string ToString() {
3     return "Name: ." + Name +"Age: " + Age;
4 }
5 //Sử dụng string.Format
6 public string ToString() {
7     return string.Format("Name: {0}.Age: {1}", Name, Age);
8 }
9 
```

Việc cộng chuỗi rất dễ nhầm lẫn và sai sót. Sử dụng **string.Format** đỡ sai sót hơn, nhưng đôi khi ta đặt nhầm vị trí các parameter thì sẽ cho kết quả sai. C# 6 đã giới thiệu một cách format string mới với dấu \$.

```
1 public string ToString() {
2     return $"Name: {Name}.Age: {Age}";
3 }
4 //Nếu muốn format chữ và số,
5 //ta chỉ thêm các kí tự format sau dấu :
6 public string ToString() {
7     return $"Name: {Name}.Age: {Age:0}";
8 }
9 
```

Rất hay và dễ đọc/dễ hiểu phải không nào.

7. Toán tử điều kiện null (?.)

Theo mình, đây là cải tiến hay nhất của phiên bản này. Lập trình lâu năm, hẳn bạn nào cũng từng đau đầu vì “*NullPointerException*“. Để tránh bị Exception này, ta thường phải check null rất nhiều lần, làm code dài dòng như sau

```
1 if (response != null && response.Result != null
2     && response.Result.Status == Status.Success) {
3     Console.WriteLine("Success");
4 }
```

(Các bạn nên xem lại khái niệm [short-circuited](#) để biết vì sao đoạn code trên chạy đúng nhé)

Với toán tử ?. mới: nếu đối tượng là null, câu lệnh sẽ trả ra null; nếu đối tượng khác null, câu lệnh sẽ chạy bình thường. Code sẽ ngắn gọn và dễ đọc hơn nhiều.

```
1 if (response?.Result?.Status == Status.Success) {
2     Console.WriteLine("Success");
3 }
```

```
4 //Cách viết cũ (Tránh NullPointerException)
5 if (stream != null) stream.Close();
6 //Cách viết mới
7 stream?.Close();
8
9 //Vd class Student có property Age kiểu int
10 //student?.Age sẽ trả về kiểu int?
11 //Bạn nên kết hợp với toán tử null (??)
12
13 //Lấy tuổi của student, nếu null thì studentAge = 0
14 int studentAge = student?.Age ?? 0;
15
```

Lưu ý nho nhỏ là chỉ có phiên bản Visual Studio 2015 mới hỗ trợ cú pháp C# 6.0 nhẹ
Mình tạo project mới trong VS2015, dùng VS2013 vẫn mở được nhưng nó không
hiểu các cú pháp mới :'(.

Bài viết có tham khảo từ bản gốc tiếng

Anh: <http://geekswithblogs.net/BlackRabbitCoder/archive/2015/04/27/the-little-wonders-of-c-6—a-presentation-to.aspx>

Series C# hay ho: So sánh 2 object trong C# (Deep compare)

Posted on 16/06/2015 by Phạm Huy Hoàng

Lâu rồi không viết bài về technical nên phải viết 1 bài cho thiên hạ biết mình vẫn code :D. Ở bài viết này, mình sẽ nói về một chuyện khá đơn giản trong C#: So sánh 2 object. Đây là một vấn đề ai cũng tưởng là dễ, mình sẽ nâng dần vấn đề lên từ đơn giản đến phức tạp. Cách giải quyết cũng sẽ từ đơn giản trở nên phức tạp, sau đó sẽ trở lại đơn giản. Nếu chịu khó đọc bài viết này từ đầu đến cuối, các bạn sẽ ngộ ra nhiều điều, khả năng technical cũng sẽ tăng kha khá đấy.

Cấp độ 1: Class đơn giản

Chúng ta bắt đầu bài toán với một class đơn giản nhé. Class này có 3 properties, khi ta gọi hàm Equals, C# chỉ so sánh reference, do đó kết quả là False

```
1 public class Student
2 {
3     public string FirstName { get; set; }
4     public string LastName { get; set; }
5     public int Age { get; set; }
6 }
7
8 var student1 = new Student
9 {
10     FirstName = "Pham",
11     LastName = "Hoang",
12     Age = 15,
13 };
14
15 var student2 = new Student
16 {
17     FirstName = "Pham",
18     LastName = "Hoang",
19     Age = 15,
20 };
21
22 student1.Equals(student2); //False
```

Để xử lý chuyện này, chúng ta chỉ cần override lại hàm Equals là xong, không phức tạp, chắc bạn nào cũng đã học ở trường nha

```
1 public override bool Equals(object obj)
2 {
```

```

3     if (obj == null || GetType() != obj.GetType()) return false;
4
5     var student = (Student) obj;
6
7     return FirstName.Equals(student.FirstName)
8         && LastName.Equals(student.LastName)
9         && Age == student.Age
10        && BirthDate.Equals(student.BirthDate);
11    }

```

Cáp độ 2: Sử dụng extension method

Hàm Equals vẫn còn khuyết điểm, đó là student1 không được null. Ta có thể xử lý vấn đề này bằng cách sử dụng extension method như sau. Giờ ta có thể gọi student1.Equals(student2) dù student1 có null đi nữa:

```

1 public static bool DeepEquals(this Student obj, Student another)
2 {
3     //Nếu null hoặc giống nhau, trả true
4     if (ReferenceEquals(obj, another)) return true;
5
6     //Nếu 1 trong 2 là null, trả false
7     if ((obj == null) || (another == null)) return false;
8
9     return obj.FirstName.Equals(another.FirstName)
10        && obj.LastName.Equals(another.LastName)
11        && obj.Age == another.Age
12        && obj.BirthDate.Equals(another.BirthDate);
13 }

```

Cáp độ 3: Viết method để so sánh 2 object nói chung

Hãy nghĩ tới trường hợp ứng dụng của bạn có khoảng vài chục class, mỗi class có vài chục property. Bạn sẽ làm gì? Hì hục viết tay hàm Equals cho từng class? Hãy sử dụng Refection trong C# để viết 1 hàm so sánh object có thể dùng cho mọi object.

```

1 public static bool DeepEquals(this object obj, object another)
2 {
3
4     if (ReferenceEquals(obj, another)) return true;
5     if ((obj == null) || (another == null)) return false;
6     //So sánh class của 2 object, nếu khác nhau thì trả fail
7     if (obj.GetType() != another.GetType()) return false;
8
9     var result = true;
10    //Lấy toàn bộ các properties của obj
11    //sau đó so sánh giá trị của từng property
12    foreach (var property in obj.GetType().GetProperties())
13    {
14        var objValue = property.GetValue(obj);
15        var anotherValue = property.GetValue(another);

```

```

14         if (!objValue.Equals(anotherValue)) result = false;
15     }
16     return result;
17 }
18 }
19
20

```

Hàm này chạy khá OK. Bạn đang tự hỏi: Ôi dào, đê thê này mà cũng viết? Chưa đâu, hãy xem tiếp phía dưới, nhiều thứ “đáng sợ” hơn đang chờ đây.

Cấp độ 4: Trường hợp object chứa object hoặc struct như DateTime

Ở cấp độ trên, ta viết hàm so sánh từng trường. Thế nhưng giả sử trong class Student, ta có chứa DateTime hoặc 1 class khác thì sao nhỉ??

```

1 public class Student
2 {
3     public string FirstName { get; set; }
4     public string LastName { get; set; }
5     public DateTime BirthDate { get; set; }
6     public Teacher Teacher { get; set; }
7 }
8
9 public class Teacher
10 {
11     public string Name { get; set; }
12     public string Subject { get; set; }
13 }
14
15 var student1 = new Student
16 {
17     FirstName = "Pham",
18     LastName = "Hoang",
19     BirthDate = new DateTime(1992, 12, 5),
20     Teacher = new Teacher { Name = "Le Minh", Subject = "Math" }
21 };
22
23 var student2 = new Student
24 {
25     FirstName = "Pham",
26     LastName = "Hoang",
27     BirthDate = new DateTime(1992, 12, 5),
28     Teacher = new Teacher { Name = "Le Minh", Subject = "Math" }
29 };

```

Suy nghĩ 1 tí nào. Để giải quyết, ta cũng sẽ so sánh từng trường, nhưng nếu trường đó là một object thì ta sẽ so sánh bằng hàm DeepEquals đã viết. Một thuật giải đệ quy cơ bản thôi mà :D

```

1 public static bool DeepEquals(this object obj, object another)
2 {
3     if (ReferenceEquals(obj, another)) return true;
4     if ((obj == null) || (another == null)) return false;
5     if (obj.GetType() != another.GetType()) return false;
6     //Nếu property không phải class, chỉ là int, double, DateTime v...v
7     //Gọi hàm equal thông thường
8     if (!obj.GetType().IsClass) return obj.Equals(another);
9
10    var result = true;
11    foreach (var property in obj.GetType().GetProperties())
12    {
13        var objValue = property.GetValue(obj);
14        var anotherValue = property.GetValue(another);
15        //Tiếp tục đệ quy
16        if (!objValue.DeepEquals(anotherValue)) result = false;
17    }
18    return result;
19}
20

```

Wow, thê là đã xong, mọi vấn đề đã được giải quyết, bạn đang tự khen mình giỏi. Ô, thê còn trường hợp không phải một object, mà là một List thì sao nhỉ, cảng đây :(

Cấp độ 5: So sánh 2 list

May thay, ta có thể viết extension method cho list như sau (Chữ <T> là generic nhé, các bạn có thể đọc bài [này](#) để xem lại).

```

1 public static bool DeepEquals<T>(this IEnumerable<T> obj, IEnumerable<T>
2 another)
{
3     if (ReferenceEquals(obj, another)) return true;
4     if ((obj == null) || (another == null)) return false;
5
6     bool result = true;
7     //Duyệt từng phần tử trong 2 list đưa vào
8     using (IEnumerator<T> enumerator1 = obj.GetEnumerator())
9     using (IEnumerator<T> enumerator2 = another.GetEnumerator())
{
10        while (true)
11        {
12            bool hasNext1 = enumerator1.MoveNext();
13            bool hasNext2 = enumerator2.MoveNext();
14
15            //Nếu có 1 list hết, hoặc 2 phần tử khác nhau, thoát khỏi vòng lặp
16            if (hasNext1 != hasNext2
17            || !enumerator1.Current.DeepEquals(enumerator2.Current))
18            {
19                result = false;
20                break;
21            }
22
23        //Đừng vòng lặp khi 2 list đều hết
24    }
25    return result;
26}
27

```

```
23         if (!hasNext1) break;
24     }
25 }
26     return result;
27 }
28 }
29 }
```

Phù, tạm xong. Chắc không còn gì nữa đâu nhỉ?

```
1 var list1 = new List<Student> { student1, student2 };
2 var list2 = new List<Student> { student1, student2};
3
4 list1 == list2; //True
```

Cấp độ 6: Một động những thứ tǎ pín lù khác

Bạn chợt nhớ ra rằng, trong C# còn hằng hà sa số những thứ tương tự List như **Dictionary**, **HashSet**,... chẳng lẽ phải viết hết. Còn một vài trường hợp tréo ngoe hơn, vd như class Student sẽ chứa 1 list cái Teacher, method chúng ta viết không chạy được

```
1 var teacherA = new Teacher { Name = "Le Minh", Subject = "Math"};
2 var teacherB = new Teacher { Name = "Tai Phu", Subject = "Physics"};
3
4 var student1 = new Student
{
5     FirstName = "Pham",
6     LastName = "Hoang",
7     Age = 15,
8     BirthDate = new DateTime(1992, 12, 5),
9     Teacher = new List<Teacher> { teacherA, teacherB}
10    };
11
12 var student2 = new Student
13 {
14     FirstName = "Pham",
15     LastName = "Hoang",
16     Age = 15,
17     BirthDate = new DateTime(1992, 12, 5),
18     Teacher = new List<Teacher> { teacherA, teacherB}
19    };
20 }
```

Tới đây mình cũng bó tay rồi, con đường phía trước khá rắc rối gian nan và phức tạp :(. Bạn được chọn 1 trong 2 lựa chọn sau:

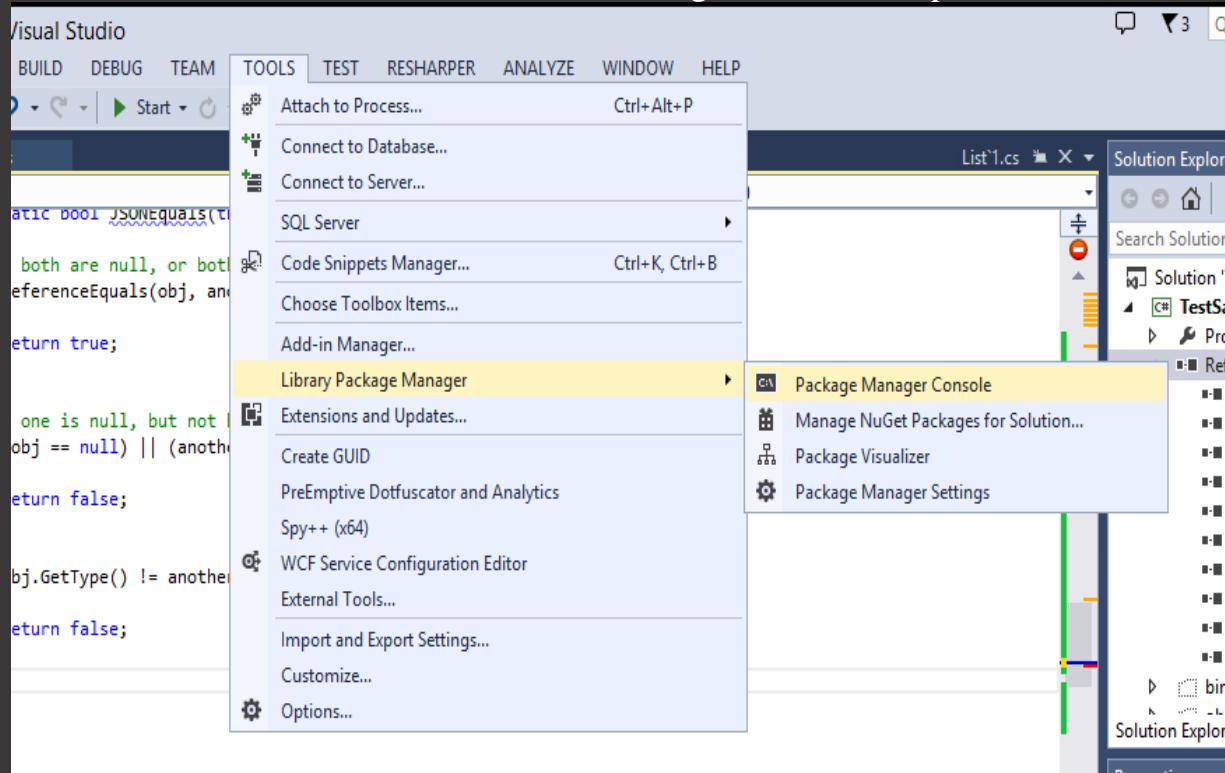
1. Với những class khó quá, hãy tự viết hàm Equals, cũng không nhiều lắm
2. Theo đuổi hàm generic tới cùng, bằng cách đọc tiếp bài này.

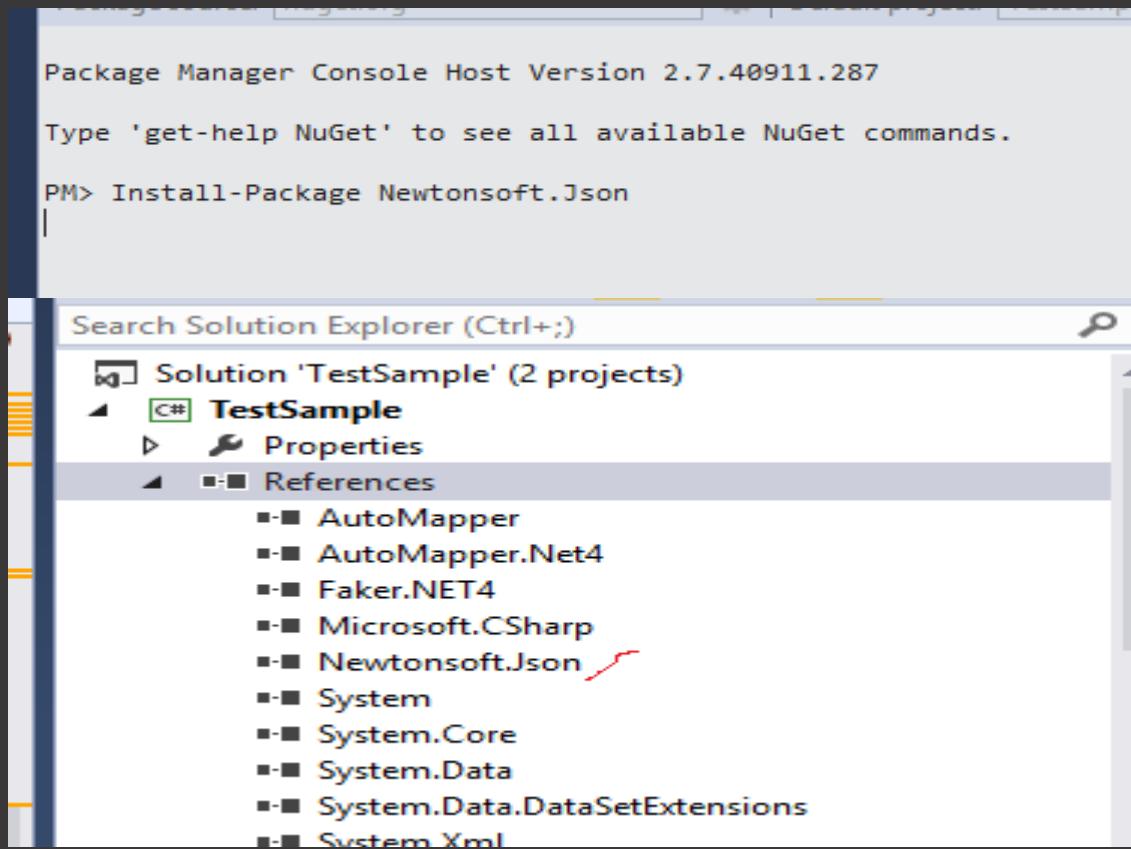
Cáp độ cuối cùng: Câu cứu anh JSON

Cách giải quyết vấn đề thật ra đơn giản hơn bạn nghĩ. Hãy serialize 2 object đó dưới dạng chuỗi JSON, so sánh 2 chuỗi được tạo ra là xong. (Việc serialize ra json đã giải quyết 99% vấn đề phức tạp liên quan đến các kiểu dữ liệu rồi, may quá :D).

Các bước thực hiện:

1. Add Reference Newtonsoft.JSON theo hướng dẫn, được kết quả như hình 3 là ok.





2. Viết 1 hàm so sánh gọn nhẹ đơn giản:

```
1 public static bool JSONEquals(this object obj, object another)
2 {
3     if (ReferenceEquals(obj, another)) return true;
4     if ((obj == null) || (another == null)) return false;
5     if (obj.GetType() != another.GetType()) return false;
6
7     var objJson = JsonConvert.SerializeObject(obj);
8     var anotherJson = JsonConvert.SerializeObject(another);
9
10    return objJson == anotherJson;
11 }
```

Có 1 vài thư viện khác, các bạn có thể search trên google với từ khóa: Deep Compare C#.

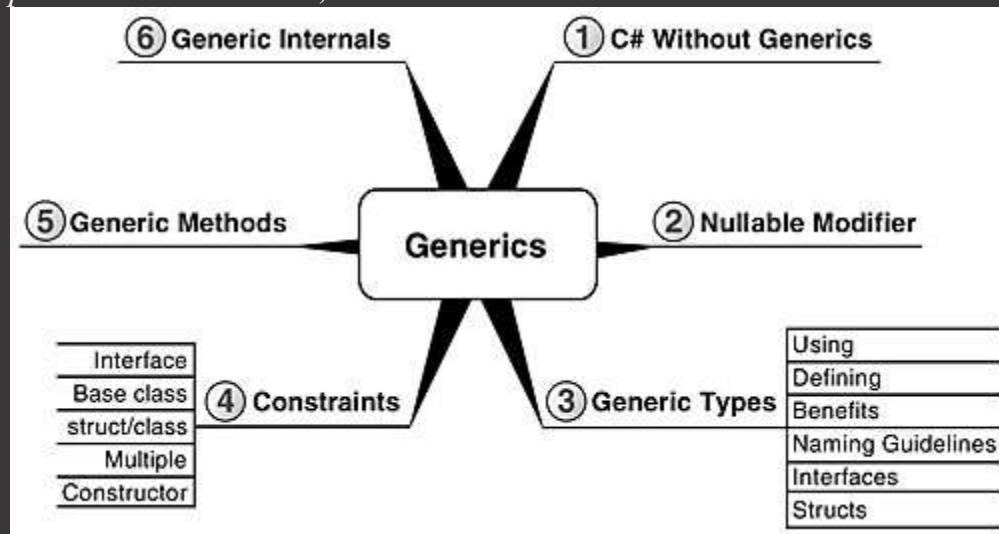
Vì bài viết khá dài, lại hơi thiên về technical nên mình đã cố gắng khiến nó hấp dẫn hơn. Chúc mừng các bạn nếu các bạn chịu khó đọc được đến cuối cùng. Phần thưởng cho các bạn kiên nhẫn đây: 5 bạn comment đầu tiên trong bài viết này có quyền request mình viết 1 bài về 1 khía cạnh trong C#, MVC hoặc javascript mà bạn muốn tìm hiểu nhé. Chúc các bạn may mắn.

Series C# hay ho: C# đã tiến hóa như thế nào (Những thay đổi của C# từ 1.0 cho tới 5.0)

Posted on 14/05/2015 by Phạm Huy Hoàng

Chắc là gần đây, công ty mình vừa tuyển thêm 1 anh Technical Lead. Đợt mình hỏi ông leader phỏng vấn thế nào, ông nhận xét “Kiến thức base C# khá vững, nói được những thay đổi của C# từ 1.0 tới 5.0”. Mặc dù mình chỉ là junior dev nhưng mình thấy phần này không khó, do lỡ tìm hiểu rồi nên post bài này chia sẻ cho các bạn luôn. Kiến thức ở bài viết này khá có ích, thích hợp đem đi chém gió, phỏng vấn hoặc hù mem mới.

Lưu ý nhỏ: C# thường được release với .NET. C# là ngôn ngữ lập trình, còn .NET là 1 thư viện/framework (Ta có VB.NET, F#, ASP.NET v...v). Trong phạm vi bài viết, mình chỉ đề cập về những thay đổi trong bản thân ngôn ngữ C#, không giới thiệu những công nghệ mới qua từng phiên bản .NET như EF, WIF, v...v nhé. Có một số phần mình đã viết rồi, chỉ dẫn link tới bài viết cũ nhé.



1. C# 2 – Giải quyết vấn đề từ C# 1

- Kiểu dữ liệu Generic: Nói đơn giản, nó là kiểu <T> mà các bạn hay dùng ấy. Với generic, code trở nên gọn gàng sáng sủa hơn, ta cũng có thể tái sử dụng function cho nhiều kiểu dữ liệu khác nhau.
- Kiểu dữ liệu nullable: Trong nhiều trường hợp, ta cần thể hiện giá trị null với kiểu dữ liệu tham trị như int, double v...v. Với 1 số ngôn ngữ khác, ta phải chuyển nó sang

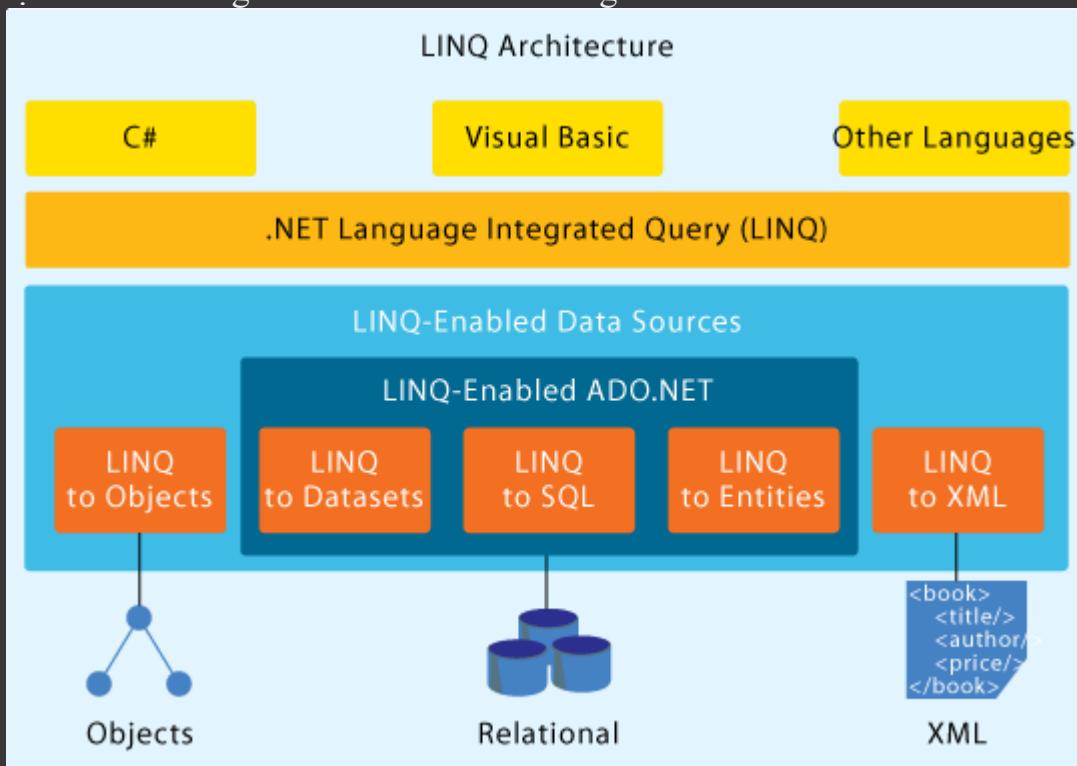
dạng tham chiếu như Intreger, Double. Với C# 2, ta có thể dùng kiểu nullable, khai báo bằng cách: Nullable<int> hoặc int?. Rất ngắn gọn và đơn giản/

c. Delegate: Đã có thể sử dụng anonymous delegate, viết ngắn học hơn so với C# 1

d. Partial class: Một partial class có thể được khai báo trong nhiều file khác nhau.

Partial class được áp dụng trong Winform, class tự sinh từ Entity Framework, giúp ta dễ dàng thêm chức năng vào form, class mà không cần động tới những class được C# tự tạo ra.

d. Iterator (IEnumerable): Để duyệt từng phần tử một mảng, ta có thể dùng hàm foreach. Iterator là một cực hình khi implement trong C# 1. Với từ khóa yield, việc tạo Iterator trong C# đã trở nên rất dễ ràng

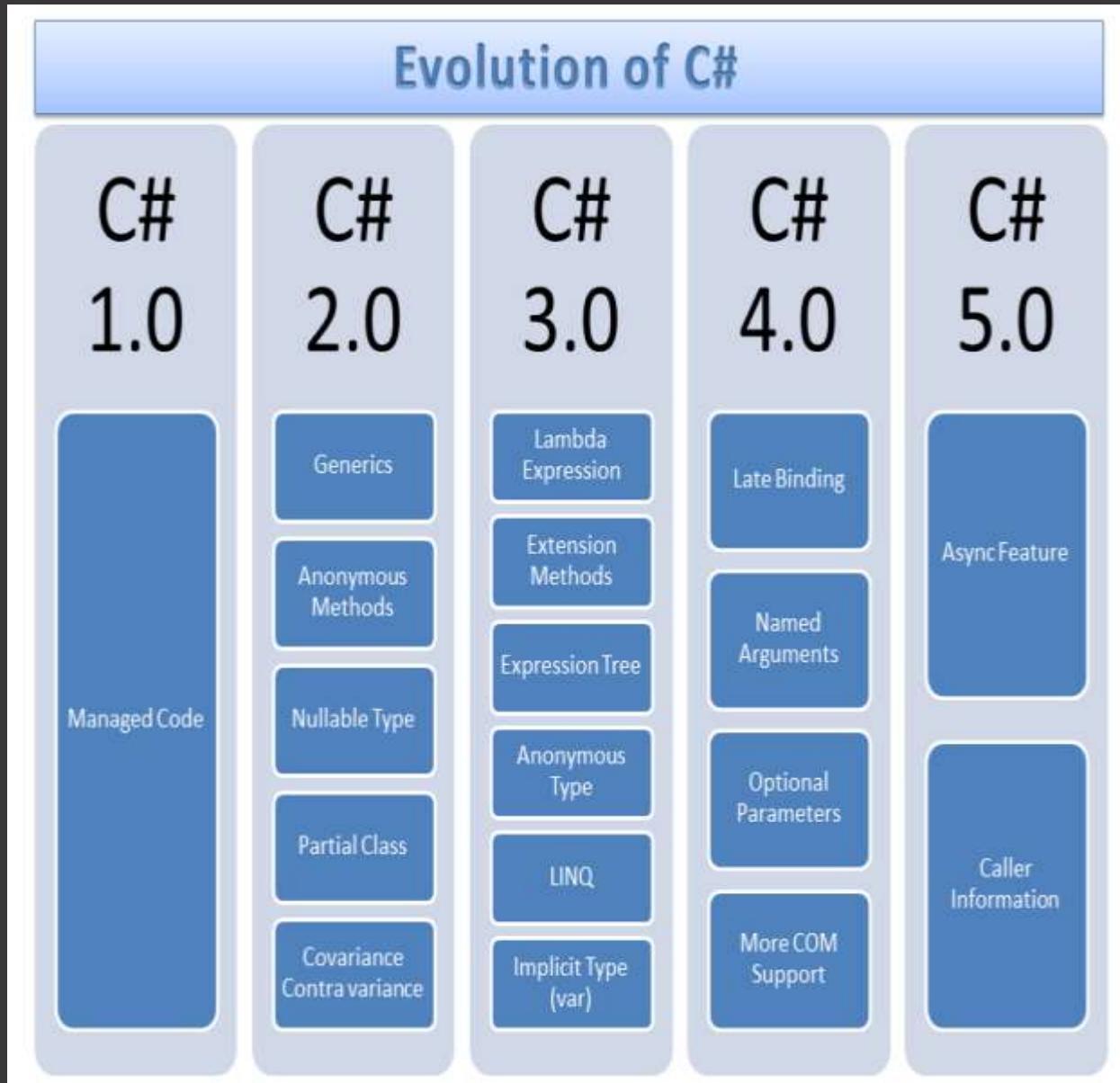


2. C# 3 – Cải tiến trong data access, LINQ chào đời

a. Một số thay đổi nhỏ trong compiler: Một số thay đổi nhỏ được C# thêm vào, giúp việc code trở nên nhẹ nhàng hơn, bạn có thể xem code của các thay đổi này trong 2 phần đầu của series C# hay ho.

- Tự implement getter, setter thông qua properties
- Khởi tạo object và array một cách dễ dàng

- Khai báo bằng biến var
 - Anonymous class
- b. Lambda expression: Lambda expression giúp việc viết delegate trở nên dễ dàng hơn rất nhiều. Mình đã viết 1 bài khá kĩ về thứ này, bạn có thể xem lại nếu quên.
- c. Extension method: Bạn đọc lại phần mình viết về extension method ở bài viết trong link nhé.
- d. Query và LINQ: Đây là phiên bản C# đầu tiên **LINQ ra đời, đã làm vỡ lâm dậy sóng, giang hồ chấn động**. Những cải tiến như var, anonymous class, lamda expression, extension method đều là nền tảng để xây dựng LINQ.
- e. Ứng dụng khác của LINQ: Ban đầu, LINQ được dùng để filter, sort v...v các object trong bộ nhớ. Nhờ sự trợ giúp của expression tree, LINQ đã có 1 số ứng dụng khác như: LINQ to SQL (Dịch lambda expression thành SQL, query trực tiếp dưới database), LINQ to XML, Parellel LINQ.



3. C# 4 – Thay đổi nhỏ nhở

- a. Cải tiến nhỏ giúp code ngắn gọn hơn:Thêm named arguments và optional parameters
- b. Lập trình với dynamic: Đây là thay đổi lớn nhất và đáng chú ý nhất trong C#. Với dynamic, giờ đây ta có thể làm việc với các object từ những ngôn ngữ dynamic như Python, Ruby, Javascript, ... Nghe có vẻ hoành tráng nhưng mình thấy dynamic ít được xài trong cái project thật, không hiểu vì sao :(.

4. C# 5 – Giảm thiểu đau đớn khi code bất đồng bộ

Điểm nhảm lớn nhất của phiên bản C# 5 này chính là 2 từ khóa async và await. Nhiều function trong C# đã được hỗ trợ call async và await. Phần này thì mình xin thú thật là mình cũng đọc rồi, nhưng do chưa áp dụng nhiều nên không hiểu mấy, có lẽ khi mình hiểu sâu hơn về vấn đề này, mình sẽ viết 1 bài hướng dẫn. (Ngày xưa máy cái như event, delegate mình cũng ngu vậy áy, về sau gặp trong code nhiều là tự động khá lên à.)

Cảm ơn sự quan tâm theo dõi của các bạn đọc trong thời gian qua. Trong 4 tháng mình đạt được mốc 1000 view, nhưng mới quảng cáo trên fb 1 ngày đã được hơn 1000 view, ngày hôm sau được ~400 view nữa :'(, sức mạnh của quảng cáo thật là to lớn quá a :'(.

C# là ngôn ngữ tuyệt vời nhất. Java, PHP, C, C++, Ruby chỉ toàn là thứ rẻ tiền

Posted on 30/07/2015 by Phạm Huy Hoàng

Sau một khoảng thời gian dài lập trình, mình đã có thể tự mà phát biểu rằng: **C# là ngôn ngữ lập trình tuyệt vời và đáng học nhất.** Lý do ư, vô số kẽ:

- Bản thân ngôn ngữ C# có vô số điều thú vị: static method, partial class, delegate, LINQ, lambda expression, ... Cái ngôn ngữ cùi bắp như Java làm gì có partial class, delegate, đến Java 8 mới bắt chước được cái lambda expression còn gì.
- C# là ngôn ngữ strong-typed: Các tham số, kết quả trả về của hàm đều là một object. Mọi lỗi do đánh nhầm tên trường, tên hàm, nhầm kiểu class đều được báo trong lúc viết code, không phải chờ đến lúc chạy mới báo như mấy cái ngôn ngữ PHP, python cùi chuối khác.
- C# đi kèm với framework .NET, hỗ trợ nhiều thứ: Tạo ứng dụng Window với WinForm, WPF; Tạo website bằng WebForm, MVC.NET... Mấy cái ngôn ngữ cấp thấp như C, C++ tuổi gì làm được mấy cái đó.
- C# có IDE Visual Studio cùng nhiều plug-in vô cùng mạnh mẽ. VS ra bản mới đều đặn như FIFA. Reshaper hỗ trợ refactor, tăng tốc độ code ... Mấy đứa khác code PHP, Python dùng cái gì để code? Dĩ nhiên là mấy thứ rởm rởm như Notepad++ hoặc Sublime Text rồi, đến cái chức năng “Jump to Definition” còn không có.

Đọc xong đoạn này, có lẽ sẽ có khoảng vài chục người ném cà chua, trứng thối và gạch đá đủ cho mình xây biệt thự. Từ từ, ít ra hãy bỏ thời gian kéo xuống dưới, đọc hết bài viết rồi ném gạch mình nhé. Dù sao khung comment nó nằm tận dưới cuối trang mà.

Chúng ta đang xem ngôn ngữ lập trình như một thú tôn giáo

Ngày xưa, mình cũng hay nhảy vào ném gạch khi nghe có đứa mở mồm chê C# và .NET. Giữa lập trình viên với nhau luôn có những cuộc tranh cãi liên tu bất tận về ngôn ngữ và công nghệ: Ngôn ngữ nào mạnh nhất, công nghệ nào tốt nhất. Ngôn ngữ, thứ vốn chỉ là công cụ, nay **được nâng lên tầm TÔN GIÁO**. Lập trình viên chia thành **đạo Java, đạo PHP, đạo C#, đạo** này công kích chửi bới đạo kia. Mức độ cuồng tín đôi khi chắc cũng không thua fan bóng đá, fan cuồng K-pop hay ISIS.

Những cuộc cãi vã chê bai đầy rẫy trên mạng, các bạn có thể thử google: *Why C# sucks*, *Why Java sucks*, *Why PHP sucks*, ... để xem thử.

Khi làm việc nhiều với một ngôn ngữ, một developer sẽ quen dần với ngôn ngữ đó, tìm ra được nhiều điều hay ho ẩn trong ngôn ngữ. Nhiều người sẽ nghĩ rằng ngôn ngữ của mình là nhất, có thể giải quyết được mọi vấn đề (Giống như ISIS nghĩ rằng đạo Hồi là nhất, mọi lời nói của đấng tối cao đều đúng đắn). Khi ngôn ngữ mình thích bị chê bai, bị xúc phạm, họ cảm thấy như chính tôn giáo của mình bị xúc phạm. Họ xù lông lên, kêu gọi bạn bè, đồng đội cùng đạo, nhảy vào ném đá cho chết “cái thằng bồ láo, dám chê Java, PHP, C++, ... của bố”.

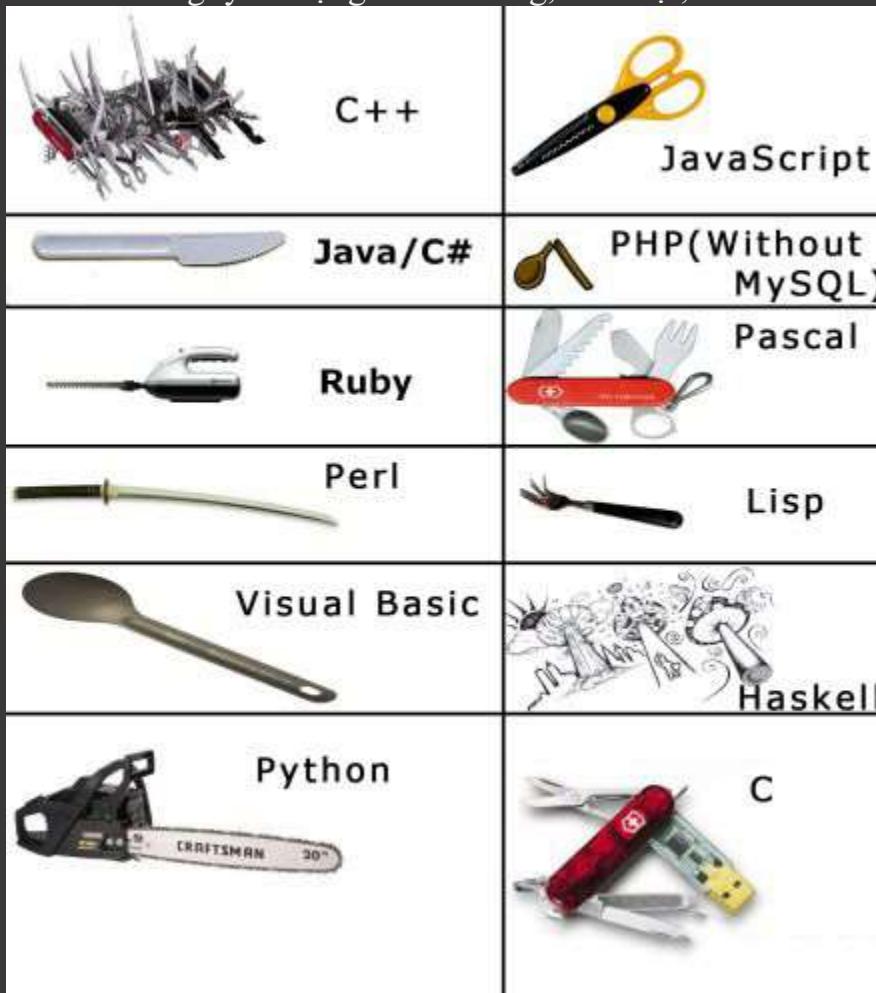


Về bản chất, ngôn ngữ chỉ là công cụ

Ngôn ngữ chỉ là thứ chúng ta sử dụng, nó không định hình nên con người chúng ta. Để mở rộng tầm nhìn, bạn hãy thử tìm hiểu nhiều ngôn ngữ xem. Bạn sẽ ngạc nhiên khi thấy giữa chúng đều có một vài khái niệm, khuôn mẫu chung. (Mình từng dùng *MVC.NET*, *Struts2*, *Django*, 3 framework của 3 ngôn ngữ khác nhau nhưng đều dựa trên khái niệm MVC cả). Nói một cách công bằng, ngôn ngữ nào cũng có cái hay của nó:

- C, C++ làm web khá cực và mất thời gian, nhưng để lập trình nhúng, lập trình game hay cần performance thì khó ai bằng nó.
- JavaScript là cái ngôn ngữ kì dị điên khùng và cực tệ. Tuy nhiên do có vô số framework đi kèm nên hiện tại và tương lai nó vẫn sẽ hot, do đó mình khuyên nhiều ban nên học.

- PHP được thiết kế dở tệ (Vốn nó được tạo ra chỉ để viết mấy trang web nhỏ), nhưng có vô số framework, cộng đồng lập trình viên đông và hung hăn. Nó là lựa chọn số 1 nếu muốn tạo 1 trang web nhanh, nhiều tính năng, ít lỗi (Điển hình như blog này viết trên wordpress, cũng viết bằng PHP).
- C#.NET, muốn dùng phải cài 1 đóng thứ nặng nề và **tốn tiền**. Nhưng nó lại được rất nhiều công ty sử dụng vì tính năng, bảo mật, v...v

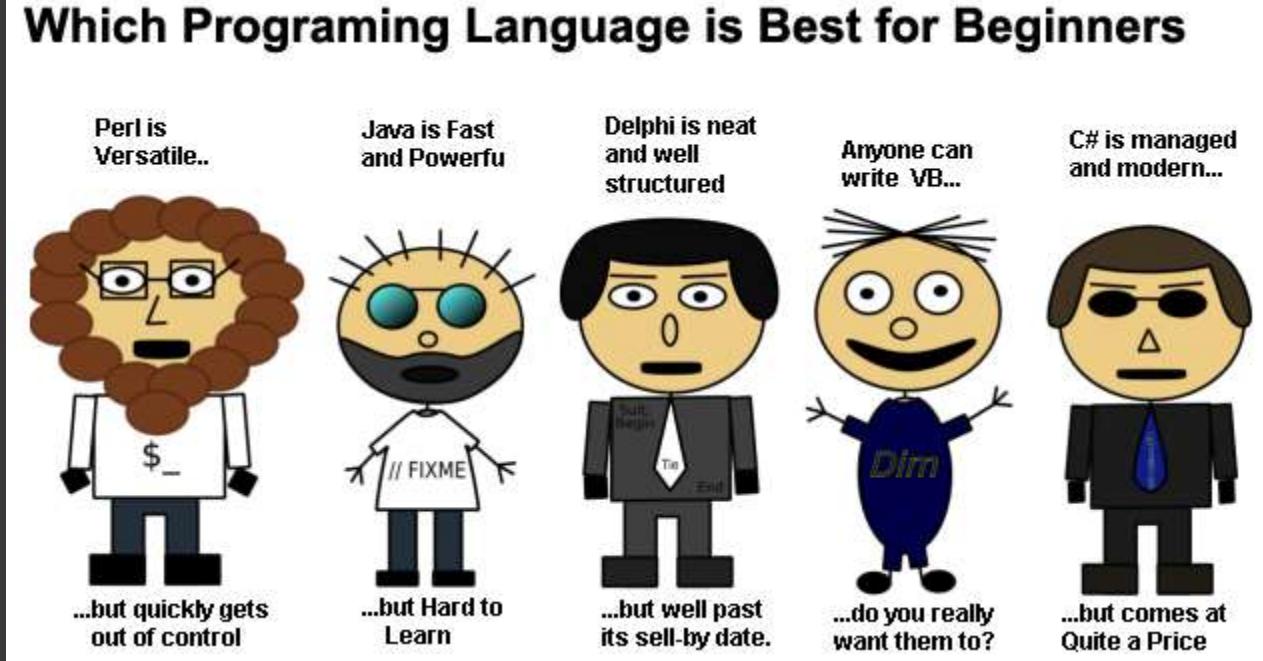


Dùng tranh cãi lại, bót gạch đá đi

Xét cho cùng, thứ quan trọng không phải là ngôn ngữ, mà là khả năng **tư duy logic**, kĩ năng **giải quyết vấn đề, tầm nhìn hệ thống**. Khách hàng sẽ đánh giá chúng ta qua sản phẩm – thứ họ thấy, và éo ai quan tâm đến code ban viết đâu. Bạn có ngừng dùng facebook vì nó viết bằng PHP – thứ ngôn ngữ cùi bắp không? KHÔNG. Bạn có bỏ stackoverflow khi biết nó được xây dựng dựa trên MVC.NET, ngôn ngữ vừa chậm vừa mắc tiền không? DĨ NHIÊN LÀ KHÔNG. Vậy thì hãy đánh giá một lập trình viên qua thứ họ làm ra, chứ đừng thông qua ngôn ngữ họ sử dụng.

Thay vì chê bai, tranh cãi khi có người chê ngôn ngữ mình thích, hãy bỏ thời gian ra tìm hiểu và chia sẻ kiến thức (Bằng cách viết blog như [mình](#) này). Giữ một cái nhìn khách quan về ngôn ngữ lập trình, bạn sẽ dễ dàng thăng tiến, tìm việc hơn (Đang làm Java nhảy qua Python cũng không sao). Ngày xưa mình cũng ghét PHP lắm, sau khi tự học nó lại thấy nó có kha khá thú vị đấy chứ.

Which Programming Language is Best for Beginners



Kết luận: Nói gì thì nói, bản chất PHP vẫn là một ngôn ngữ sida cùi bắp, và Javascript vẫn là cái thứ dị hợm, dở dở ương ương, thất bại của tạo hóa. Nếu bạn vẫn còn cay cú vì PHP bị nói xấu, vui lòng kéo lên đầu và đọc lại bài viết nhé :D.

Review sách: The Passionate Programmer – Những điều giúp developer phát triển sự nghiệp

Posted on 17/03/2015 by Phạm Huy Hoàng

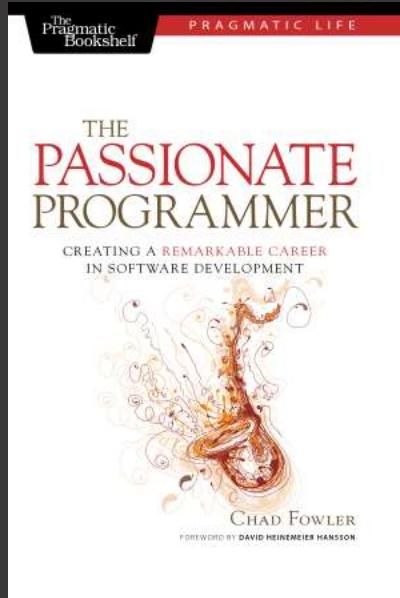
Sau một loạt những bài viết về technical khô khan, hôm nay mình sẽ đổi gió bằng 1 bài review sách. Đây là một cuốn sách nói về những điều developer cần làm để phát triển bản thân và sự nghiệp, đúng với tôn chỉ của blog mình (Lập trình viên cần biết nhiều điều hơn ngoài code). Bài này mình dùng giọng văn nhẹ nhàng hài hước nhẹ, con người thật của mình ngoài đời cũng nhẹ nhàng hài hước như vậy =)).



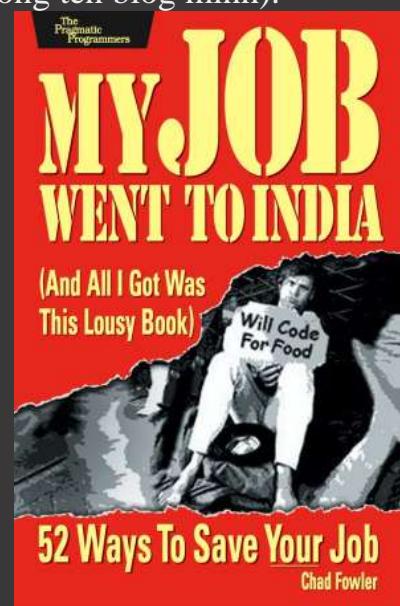
Vào một ngày làm việc bình thường như bao ngày khác, mình đi làm (Dĩ nhiên, developer nhận lương mỗi tháng thì hằng ngày phải đi làm). Như thường lệ, mình bật máy, login vào Window, bật IDE lên, check mail hôm trước. 30 phút đầu trong ngày làm việc là thời gian để thư giãn. Mình bắt đầu vào các trang quen thuộc: stackoverflow, codeproject, simpleprogrammer, ... thấy lão John Sonmez (Tác giả blog simpleprogrammer.com) đang giới thiệu cuốn Soft Skills lão vừa viết. Mình vừa nhủ thầm: Góm, sách mình cũng viết được, vừa xuất bản 1 cuốn thời còn là sinh viên, bán chạy như tôm tươi đây thôi ([Bằng chứng](#)) , vừa tìm ebook cuốn Soft Skills mà không thấy (bố khỉ).

Đọc 1 hồi, thấy lão Sonmez có nhắc tới cuốn [The Passionate Programmer: Creating a Remarkable Career in Software Development](#), bảo rằng giá như lão đọc được

cuốn dày sorm hơn. Tò mò nên mình cũng google tìm ebook đọc thử, tính mình ham đọc sách mà lại. Kết quả là các bạn có được bài review sách này đây. Đầu tiên, mình xin giới thiệu cái bìa sách:



Bìa cuốn sách được cấu thành từ 3 gam màu chính là đỏ, đen trắng, với nét vẽ trừu tượng hóa chiếc kèn clarinet Chắc không ai quan tâm tới cái này nên bỏ qua =)). Thật ra cuốn sách này là bản tái bản lần 2, lần đầu tái bản cuốn sách có tên là: Công việc của em đã chạy qua Ấn Độ (Còn lại mình em với cuốn sách cùi bắp này) : 52 cách giữ việc làm. Bìa sách gồm 3 gam màu, đen đỏ trắng, hình 1 người đàn ông **code** **đạo**(Đúng nghĩa **code** **đạo** trong tên blog mình).



Tác giả [Chad Fowler](#) là một người khá nổi tiếng, ông là người đã xây dựng nền móng cho framework [Ruby on Rails](#) huyền thoại. Huyên thuyên nay giờ đủ rồi, giờ mình sẽ chuyển qua 2 phần quan trọng nhất, sơ lược nội dung và nhận xét, đánh giá

Sơ lược nội dung sách

Xin nhắc lại: Đây là một cuốn sách viết về con đường phát triển sự nghiệp cho developer nói chung, cũng như dân IT nói riêng. Tác giả xem sự nghiệp IT như 1 sản phẩm hàng hóa. Để có một sản phẩm thành công, các doanh nghiệp thường thực hiện 4 bước: Nghiên cứu thị trường, đầu tư sản phẩm, phát triển sản phẩm, marketing. 4 bước này tương đương với 4 chương đầu của cuốn sách:

1. **Nghiên cứu thị trường:** Thị trường ở đây chính là thị trường nghề nghiệp. Sách hướng dẫn ta cách đánh giá thị trường, ảnh hưởng của quy luật cung cầu đến tiền lương, cũng như hướng dẫn cách chọn công nghệ, ngôn ngữ ta cần đầu tư.
2. **Đầu tư sản phẩm:** Sau phần 1, ta đã chọn được 1 hoặc 1 vài công nghệ để đầu tư thời gian và công sức nghiên cứu. Ở phần này, tác giả làm rõ hơn về những chuyện ta cần làm để phát triển bản thân: Ta nên học/tiếp cận 1 công nghệ ra sao? Cần làm những gì để có kết quả tốt nhất.
3. **Phát triển sản phẩm:** Phần này lại đề cập tới những điều **tối cần thiết** trong công việc. Hãy nhớ: Người khác (Đồng nghiệp, quản lý,) đánh giá bạn qua những gì **bạn làm được**, không phải qua kiến thức (vay mượn đâu đó) của bạn. Phần này nói về thái độ cũng như cách làm việc mà 1 lập trình viên nên có.
4. **Marketing:** Đã qua rồi cái thời “Hữu xạ tự nhiên hương”, bạn cứ làm tốt việc được giao và mọi người sẽ để ý tới. Đây là thời đại của Marketing, bạn có khả năng, nhưng bạn phải chứng minh **khả năng của mình**. Đây là phần hay nhất quyển sách, hướng dẫn bạn cách xây dựng hình ảnh bản thân, cũng như những cách làm cho tiếng nói của mình có trọng lượng hơn.



Nhận xét

- Phần tóm tắt có vẻ khá khô khan, tuy nhiên cuốn sách lại rất cuốn hút, đọc không hề chán hay buồn ngủ.
- Mỗi phần được chia làm 8, 9 chương, bao gồm những trải nghiệm và chia sẻ mà tác giả đã trải qua trong nghề, không hề mang tính chất dạy đòn mà mang tính chất chia sẻ.

- Cuối mỗi chương còn có 1 phần review nhỏ, tóm tắt lại những việc chúng ta nên làm sau khi đọc xong chương đó.
- Kết luận: Một cuốn sách khá hay, cách viết thú vị, dễ đọc, cũng như vô cùng hữu dụng cho những bạn lập trình viên đang cảm thấy phân vân, cần những lời khuyên trong sự nghiệp như mình. **Đánh giá: 8.5/10**

Tạo dummy data với Faker và Mockaroo – Xa rời những ngày nhập tay nhảm chán

Posted on 19/03/2015 by Phạm Huy Hoàng

Cuộc đời một thằng developer có rất nhiều việc rất chán nhưng phải làm: **fix bug, viết report, nhập timesheet, viết code test** ... Một trong những việc đáng chán đó là: **Tạo data giả để hiển thị và test**. Để làm việc này, chúng ta thường tạo object giả (bằng code), hoặc đánh data giả vào SQL. Hầu như 90% developer và tester đều ghét cái việc vừa nhảm nhí vừa đáng chán này, do dó ta thấy dữ liệu trong data giả thường là: test1,test2, testemail@mail.com....

Cũng phải nói thêm, ngoài nhảm chán, công việc này còn khá mất thời gian. Hãy tưởng tượng ta có 5 bảng, mỗi bảng 10 cột, mỗi cột cần điền 10-20 dòng data giả, mấy thời gian quá phải không nào? Giờ đây, với Faker và Mockaroo, ta có thể **tạo data giả theo phong cách developer**, nhanh gọn, thú vị và không tốn thời gian.

1. Tạo data giả trong javascript

Khi sử dụng một số java script framework, ta cần 1 số data giả dưới dạng java script object để hiển thị. Ta có thể sử dụng faker.js cho mục đích trên.

Các bạn tải faker.js tại đây (Bấm DownloadZIP đây nhé)

<https://github.com/marak/Faker.js/>

Giải nén ra, trong thư mục build lấy file faker.js. Thêm reference tới file đó như sau:

```
1 <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
2   <head>
3     <script src="faker.js" type="text/javascript"></script>
4     <title>Test Random</title>
5   </head>
6
7   <body></body>
8 </html>
```

Việc sử dụng code khá đơn giản, các bạn chỉ cần sử dụng các function trong class faker. Tham khảo thêm API ở đây: <http://marak.com/faker.js/>

Dưới đây là code mẫu mình tạo 100 object của class Student

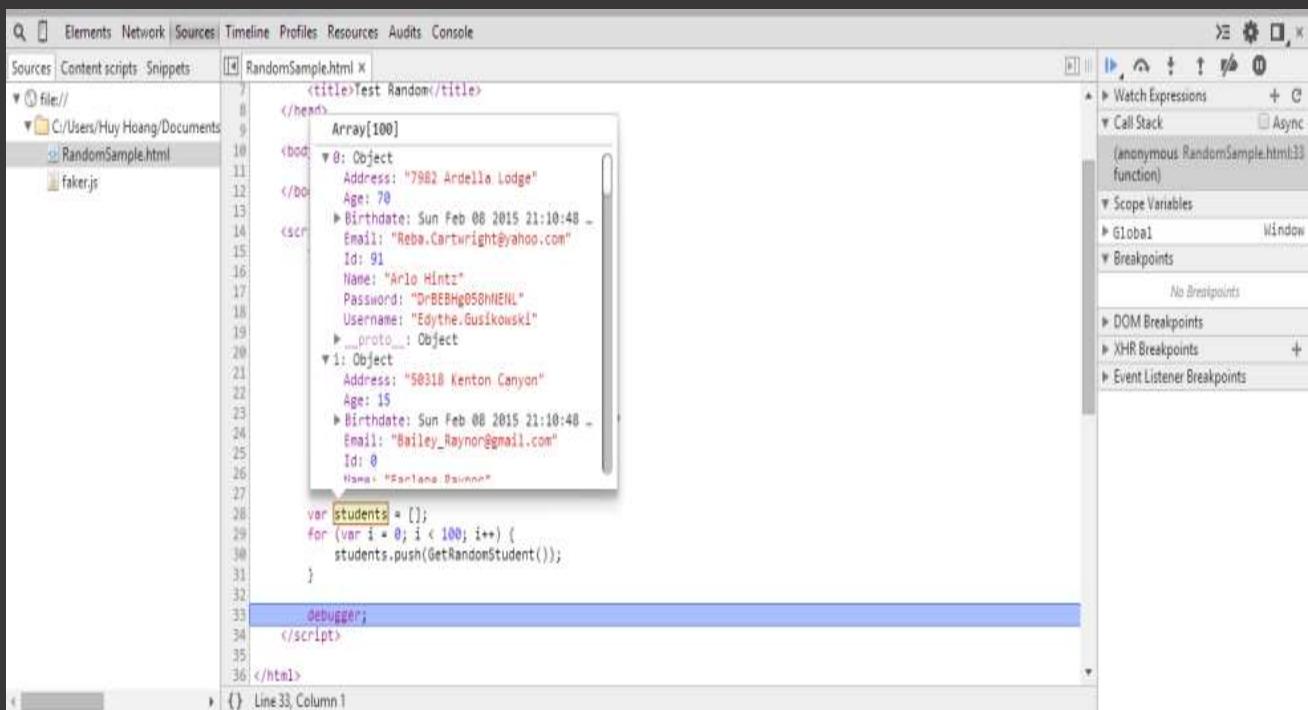
```
1 function GetRandomStudent() {
2   return {
```

```

3     Id : faker.random.number(100),
4     Name : faker.name.findName(),
5     Username : faker.internet.userName(),
6     Password: faker.internet.password(),
7     Email : faker.internet.email(),
8     Age : faker.random.number(100),
9     Address: faker.address.streetAddress(),
10    Birthdate: faker.date.past()
11  };
12 }
13
14 var students = [];
15 for (var i = 0; i < 100; i++) {
16   students.push(GetRandomStudent());
17 }

```

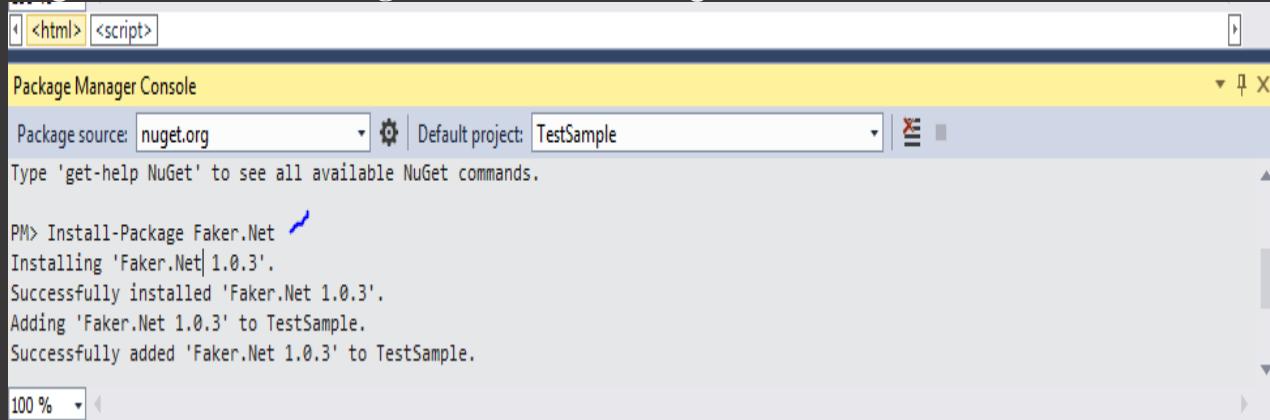
Và đây là kết quả. Data “giả” nhưng trông khá giống thật phải không



2. Tạo data giả trong C#

Một vài trường hợp khác, ta cần tạo data giả ở server side. Ở đây mình sẽ lấy C# làm ví dụ, với thư viện **Faker.NET** (Các ngôn ngữ khác như Java, Ruby, ... có thể tìm thư viện tương ứng).

Các bạn sử dụng Nuget để cài đặt Faker.NET. Mở Nuget Package Manager Console và gõ vào “Install-Package Faker.Net” là xong.



```
<html> <script>
Package Manager Console
Package source: nuget.org Default project: TestSample
Type 'get-help NuGet' to see all available NuGet commands.

PM> Install-Package Faker.Net
Installing 'Faker.Net 1.0.3'.
Successfully installed 'Faker.Net 1.0.3'.
Adding 'Faker.Net 1.0.3' to TestSample.
Successfully added 'Faker.Net 1.0.3' to TestSample.

100 %
```

Cách sử dụng cũng tương tự như phiên bản javascript. Có điều thư viện này chưa hỗ trợ password và datetime.

```
1 public class Student
2 {
3     public int Id { get; set; }
4     public string Name { get; set; }
5     public string Address { get; set; }
6     public string Username { get; set; }
7     public int Age { get; set; }
8     public string Email { get; set; }
9
10    public static Student GetRandomStudent()
11    {
12        var name = Faker.Name.FullName(NameFormats.Standard);
13        return new Student
14        {
15            Id = Faker.RandomNumber.Next(100),
16            Name = name,
17            Username = Faker.Internet.UserName(name),
18            Email = Faker.Internet.Email(name),
19            Age = Faker.RandomNumber.Next(50),
20            Address = Faker.Address.StreetAddress(),
21        };
22    }
23}
24
25 //Tạo 100 random student
26 var students = new List<Student>();
```

```
27 for (int i = 0; i < 100; i++)
    students.Add(Student.GetRandomStudent());
```

Và đây là kết quả

Locals		
Name	Value	Type
args	{string[0]}	string[]
students	Count = 150	System.Collections.Generic.List<TestSample.Student>
[0]	{TestSample.Student}	TestSample.Student
Address	"0126 Matt Mount"	string
Age	13	int
Email	"sunny.johns@wunsch.us"	string
Id	74	int
Name	"Sunny Johns"	string
Username	"sunny_johns"	string
[1]	{TestSample.Student}	TestSample.Student
Address	"8786 Herminio Dam"	string
Age	36	int
Email	"marcelina.krajcik@wisozk.biz"	string
Id	35	int
Name	"Marcelina Krajcik"	string
Username	"marcelina_krajcik"	string
[2]	{TestSample.Student}	TestSample.Student
Address	"145 Clotilde Stream"	string
Age	32	int
Email	"wade_walsh@wolf.info"	string
Id	34	int
Name	"Wade Walsh"	string
Username	"wade.walsh"	string
[3]	{TestSample.Student}	TestSample.Student
[4]	{TestSample.Student}	TestSample.Student
[5]	{TestSample.Student}	TestSample.Student
[6]	{TestSample.Student}	TestSample.Student
[7]	{TestSample.Student}	TestSample.Student

3. Tạo data giả dưới database

Đây là trường hợp hay gặp nhất, chúng ta cần nhập data giả vào database. Mình để nó dưới cùng để các bạn chịu khó đọc 2 cái trên, xin thứ lỗi. Ở đây, chúng ta sẽ sử dụng trang web Mockaroo để tạo data giả: <https://www.mockaroo.com/>

Trang web hỗ trợ rất nhiều kiểu data giả như sau

Choose a Type

Filter

All (86) Basic (22) IT (10) Personal (18) Location (14) Money (7) Health (10) Advanced (5)

Bitcoin Address	Blank	Boolean
1EZ5PdVcsVEaaKYH37t8toLodjc97eooy6 1HpzKTCbcJ57vkoQZCfhjyvZK97MRzktVk 1BPny1b66pdZuZC87tjikS8EG5CHRYp4ni	<i>Always generates a null value</i>	true false
City	Color	Company Name
New York Berlin London	Red Blue Black	Google Home Depot General Electric
Country	Country Code	Credit Card #
Germany France Japan	ES GR FR	4017959045824 5349690971837932 372301294706546
Credit Card Type	Currency	Currency Code
visa	Dollar	USD

Can't find what you're looking for? You can mock almost anything using Regular Expressions... [Close](#)

Giao diện cũng không khó sử dụng. Mình sẽ dùng nó để tạo 50 students cho bảng Student như sau:

Field Name	Type	Options
Id	Row Number	blank: 0 % ×
Name	Full Name	blank: 0 % ×
Username	Username	blank: 0 % ×
Password	Password	blank: 0 % ×
Email	Email Address	blank: 0 % ×
Age	Number	min: 1 max: 100 decimals: 0 blank: 0 % ×
Address	Street Address	blank: 50 % × 50% số dòng là null
Birthdate	Date	2/8/2014 to 2/8/2015 in dd/mm/yyyy blank: 0 % ×

[Add another field](#) [Thêm trường mới](#)

Số dòng

Rows: Format: Table Name: include create table SQL

Khá dễ phải không? Sau khi bấm Generate Data, bạn sẽ được tải về 1 file sql như sau, bạn chỉ cần copy file này vào SQL Server, chạy là xong. Trang web còn có thể export ra dữ liệu dưới dạng JSON và 1 số dạng khác.

nts.sql - HOAN...SS.master (sa (51)) X

```
Age, Address, Birthdate) values (1, 'Larry Fuller', 'lfuller0', 'I5Y3PHKkiaK', 'lfuller0@aboutads.info', 49, null, '23/10/2014');
Age, Address, Birthdate) values (2, 'Todd Grant', 'tgrant1', 'pfzIZ7n', 'tgrant1@sphinn.com', 68, null, '07/01/2015');
Age, Address, Birthdate) values (3, 'Wanda Campbell', 'wcampbell2', '8zQ020mYA', 'wcampbell2@fastcompany.com', 95, '78622 Heffernan');
Age, Address, Birthdate) values (4, 'Nicole Tucker', 'ntucker3', 'j32kruaxF', 'ntucker3@bizjournals.com', 79, null, '18/03/2014');
Age, Address, Birthdate) values (5, 'Nancy Franklin', 'nfranklin4', 'uE8Xp0iVVfJ', 'nfranklin4@blinklist.com', 40, null, '11/12/2013');
Age, Address, Birthdate) values (6, 'Mildred Murphy', 'mmurphy5', 'OUkEqC', 'mmurphy5@trellian.com', 10, '3 Northridge Pass', '09/08/2014');
Age, Address, Birthdate) values (7, 'Samuel Hudson', 'shudson6', 'ehybfffZ', 'shudson6@fda.gov', 11, '01 Marcy Point', '15/09/2014');
Age, Address, Birthdate) values (8, 'Douglas Ramirez', 'dramirez7', 'jVh1RnCIVutI', 'dramirez7@yellowpages.com', 87, '9 Huxley Street', '08/07/2014');
Age, Address, Birthdate) values (9, 'Judith Spencer', 'jspencer8', 'YfK404', 'jspencer8@arstechnica.com', 89, null, '09/03/2014');
Age, Address, Birthdate) values (10, 'Ruby Carr', 'rcarr9', 'tth9hxLeOn', 'rcarr9@comcast.net', 8, '8 Melvin Alley', '04/06/2014');
Age, Address, Birthdate) values (11, 'Tina Tucker', 'ttuckera', 'nhcaoFTC58', 'ttuckera@ucla.edu', 46, '81671 West Pass', '07/05/2014');
Age, Address, Birthdate) values (12, 'Brandon Cole', 'bcoleb', 'x0BF3eZ17U', 'bcoleb@loc.gov', 53, '36 Sutteridge Junction', '12/03/2014');
Age, Address, Birthdate) values (13, 'Carolyn Day', 'cdayc', 'VoKVHNB0', 'cdayc@over-blog.com', 54, null, '09/08/2014');
Age, Address, Birthdate) values (14, 'Martin Patterson', 'mpattersond', 'aVi169tVT', 'mpattersond@omniture.com', 18, null, '26/07/2014');
Age, Address, Birthdate) values (15, 'Karen Schmidt', 'kschmidte', 'W78cd4GA', 'kschmidte@google.com', 69, null, '31/10/2014');
Age, Address, Birthdate) values (16, 'Elizabeth Crawford', 'ecrawfordf', 'vBb2sn', 'ecrawfordf@ifeng.com', 80, null, '04/05/2014');
Age, Address, Birthdate) values (17, 'James Richardson', 'jrichardsong', 'sACxMTOIQ', 'jrichardsong@telegraph.co.uk', 79, '0 Hoepker');
Age, Address, Birthdate) values (18, 'Thomas Burton', 'tburtonh', 'ZwxBt0', 'tburtonh@opensource.org', 10, null, '29/09/2014');
Age, Address, Birthdate) values (19, 'Louise Carroll', 'lcarrolli', '7LvngYySOu', 'lcarrolli@illinois.edu', 29, '18 Iowa Avenue', '08/03/2014');
Age, Address, Birthdate) values (20, 'Patricia Martin', 'pmartinj', 'zFRkPK', 'pmartinj@livejournal.com', 30, '85 Stephen Alley', '08/03/2014');
Age, Address, Birthdate) values (21, 'Katherine Berry', 'kberryk', '9t19t2', 'kberryke@flickr.com', 95, '4702 Tony Hill', '17/06/2014');
Age, Address, Birthdate) values (22, 'Jeremy Torres', 'jtorresl', 'uUCMyr', 'jtorresl@bloglovin.com', 90, '0879 Del Mar Road', '27/07/2014');
Age, Address, Birthdate) values (23, 'Carlos Hall', 'challm', 'PQI09XS', 'challm@a8.net', 65, '7875 Novick Junction', '20/06/2014');
Age, Address, Birthdate) values (24, 'Carl Lawson', 'clawsonn', 'wbub6UgE3', 'clawsonn@deliciousdays.com', 2, '6 Crescent Oaks Road', '08/03/2014');
Age, Address, Birthdate) values (25, 'Jason Jordan', 'jjordano', 'qgMP5aYRV', 'jjordano@marketwatch.com', 48, '668 Westport Street', '08/03/2014');
Age, Address, Birthdate) values (26, 'Brian Wilson', 'bwilsonp', 'Wk3xf67vR', 'bwilsonp@ebay.com', 27, '1399 Old Shore Place', '30/03/2014');
Age, Address, Birthdate) values (27, 'Stephanie Welch', 'swelchq', 'WPKejBcBoU1', 'swelchq@gmpg.org', 20, null, '06/12/2014');
Age, Address, Birthdate) values (28, 'Matthew Ramirez', 'mraramirezr', 'n8wDaS1', 'mraramirezr@wsj.com', 49, '5681 Nova Road', '15/12/2013');
Age, Address, Birthdate) values (29, 'Kelly Peters', 'kpeterss', 'VZTbUh1', 'kpeterss@usda.gov', 89, null, '25/02/2014');
Age, Address, Birthdate) values (30, 'Patricia Woods', 'pwoodst', 'chMOWNsK00d1', 'pwoodst@google.ca', 89, '06 Crowley Pass', '17/07/2014');
Age, Address, Birthdate) values (31, 'Judy Coleman', 'jcolemanu', 'kzRXNpw0', 'jcolemanu@hugedomains.com', 88, '6 Victoria Point', '08/03/2014');
Age, Address, Birthdate) values (32, 'Sharon Barnes', 'sbarnesv', 'UzoZWt7', 'sbarnesv@bloomberg.com', 6, null, '02/04/2014');
Age, Address, Birthdate) values (33, 'Virginia Fields', 'vfieldsw', 'ZUj421', 'vfieldsw@geocities.com', 34, null, '02/09/2014');
Age, Address, Birthdate) values (34, 'Raymond Harvey', 'rharveyx', 'BB053c', 'rharveyx@topsy.com', 38, null, '13/08/2014');
Age, Address, Birthdate) values (35, 'Nicholas Rivera', 'nriveray', 'QgB8Jd1IH53D', 'nriveray@vistaprint.com', 50, '982 Bultman Place', '08/03/2014');
Age, Address, Birthdate) values (36, 'Brenda Gonzales', 'bgonzalesz', 'OPPM6P', 'bgonzalesz@jathis.com', 93, '476 Eagle Crest Place', '08/03/2014');
Age, Address, Birthdate) values (37, 'Betty Tucker', 'btucker10', 'jTv9Xv', 'btucker10@com.com', 99, null, '10/06/2014');
Age, Address, Birthdate) values (38, 'Kathleen Bishop', 'kbishop11', '4Gwq9I', 'kbishop11@kickstarter.com', 52, null, '09/12/2014');
Age, Address, Birthdate) values (39, 'Ruby Rose', 'rrose12', 'Nuu3LzeWssm', 'rrose12@unesco.org', 98, '9062 Katie Pass', '08/07/2014');
Age, Address, Birthdate) values (40, 'Todd Robinson', 'trobinson13', 'iXvRdj2', 'trobinson13@google.co.jp', 97, '83572 Meadow Ridge', '08/07/2014');
Age, Address, Birthdate) values (41, 'Elizabeth Russell', 'erussell14', '7BavlyTg', 'erussell14@imbin.com', 78, null, '27/07/2014');
```

Bài viết kết thúc ở đây, mong rằng nó sẽ giúp cho quãng đường developer của các bạn đỡ vất vả hơn. Nếu bài viết có ích cho bạn, nhớ share để cảm ơn mình nhé.

SOLID là gì – Áp dụng các nguyên lý SOLID để trở thành lập trình viên code “cứng”

Posted on 24/03/2015 by Phạm Huy Hoàng

Trong quá trình học, hầu như các bạn sinh viên đều được học một số khái niệm OOP cơ bản như sau:

- Abstraction (Tính trừu tượng)
- Encapsulation (Tính bao đóng)
- Inheritance (Tính kế thừa)
- Polymorphism (Tính đa hình)

Những khái niệm này đã được dạy khá rõ ràng, và hầu như những buổi phòng vấn nào cũng có những câu hỏi liên quan đến khái niệm này. Vì 4 khái niệm này khá cơ bản, bạn nào chưa vững có thể google để tìm hiểu thêm.

Những nguyên lý mình giới thiệu hôm nay là những nguyên lý thiết kế trong OOP. Đây là những nguyên lý được đúc kết bởi máu xương vô số developer, rút ra từ hàng ngàn dự án thành công và thất bại. Một project áp dụng những nguyên lý này sẽ có code dễ đọc, dễ test, rõ ràng hơn. Và việc quan trọng nhất là việc **maintainace code sẽ dễ hơn rất nhiều** (Ai có kinh nghiệm trong ngành IT đều biết **thời gian code chỉ chiếm 20-40%**, còn lại là thời gian để maintainance: thêm bớt chức năng và sửa lỗi). Nắm vững những nguyên lý này, đồng thời áp dụng chúng trong việc thiết kế + viết code sẽ giúp bạn tiến thêm 1 bước trên con đường thành senior nhé (1 ông senior bên FPT Software từng bảo mình thế).



Ok, 10 phút quảng cáo đã qua, bây giờ đến phần giới thiệu. SOLID tức là “cứng”, áp dụng nhiều thì bạn sẽ code “cứng”. Đùa thôi, nó là tập hợp 5 nguyên tắc sau đây:

1. Single responsibility principle
2. Open/closed principle
3. Liskov substitution principle
4. Interface segregation principle
5. Dependency inversion principle

Trong nội dung bài viết này, mình chỉ giới thiệu tổng quát để các bạn có cái nhìn tổng quan về các nguyên lý này.

1. Single responsibility principle



Single Responsibility Principle

Just because you *can* doesn't mean you *should*.

Nguyên lý đầu tiên, tương ứng với chữ S trong SOLID. Nội dung nguyên lý:

Một class chỉ nên giữ 1 trách nhiệm duy nhất (Chỉ có thể sửa đổi

class với 1 lý do duy nhất)

Để hiểu nguyên lý này, ta hãy lấy ví dụ với 1 class **vi phạm nguyên lý**. Ta có 1 class như sau

```
1 public class ReportManager ()  
2 {  
3     public void ReadDataFromDB () ;  
4     public void ProcessData () ;  
5     public void PrintReport () ;  
6 }
```

Class này giữ tới 3 trách nhiệm: Đọc dữ liệu từ DB, xử lý dữ liệu, in kết quả. Do đó, chỉ cần ta thay đổi DB, thay đổi cách xuất kết quả, ... ta sẽ phải sửa đổi class này.

Càng về sau class sẽ càng phình to ra. Theo đúng nguyên lý, ta phải tách class này ra

làm 3 class riêng. Tuy số lượng class nhiều hơn những việc sửa chữa sẽ đơn giản hơn, class ngắn hơn nên cũng ít bug hơn.

2. Open/closed principle



Open-Closed Principle

Open-chest surgery isn't needed when putting on a coat.

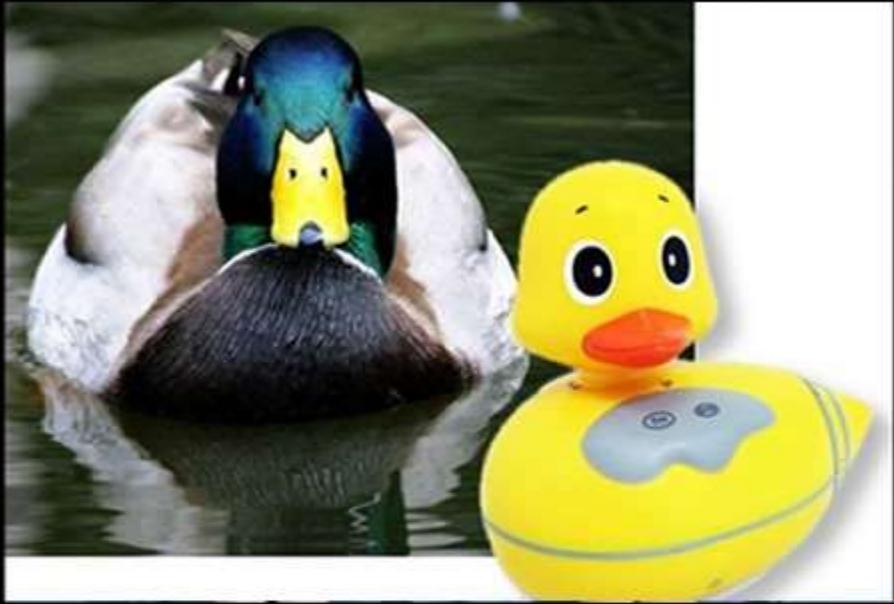
Nguyên lý thứ hai, tương ứng với chữ **O** trong SOLID. Nội dung nguyên lý:

Có thể thoái mái mở rộng 1 class, nhưng không được sửa đổi

bên trong class đó (open for extension but closed for modification).

Theo nguyên lý này, mỗi khi ta muốn thêm chức năng.. cho chương trình, chúng ta nên viết class mới mở rộng class cũ (bằng cách kế thừa hoặc sở hữu class cũ) không nên sửa đổi class cũ.

3. Liskov Substitution Principle



Liskov Substitution Principle

If it looks like a duck and quacks like a duck but needs batteries,
you probably have the wrong abstraction.

Nguyên lý thứ ba, tương ứng với chữ **L** trong SOLID. Nội dung nguyên lý:

Trong một chương trình, các object của class con có thể

thay thế class cha mà không làm thay đổi

tính đúng đắn của chương trình

Hơi khó hiểu? Không sao, lúc mới đọc mình cũng vậy. Hãy tưởng tượng bạn có 1 class cha tên **Vịt**. Các class **VịtBầu**, **VịtXiêm** có thể kế thừa class này, chương trình chạy bình thường. Tuy nhiên nếu ta viết class **VịtChạyPin**, **cần pin mới chạy được**. Khi class này kế thừa class **Vịt**, vì **không có pin không chạy được, sẽ gây lỗi**. Đó là 1 trường hợp vi phạm nguyên lý này.

4. Interface Segregation Principle



Interface Segregation Principle

You want me to plug this in *where?*

Nguyên lý thứ tư, tương ứng với chữ **I** trong SOLID. Nội dung nguyên lý:

Thay vì dùng 1 interface lớn, ta nên tách thành nhiều

interface nhỏ, với nhiều mục đích cụ thể

Nguyên lý này khá dễ hiểu. Hãy tưởng tượng chúng ta có 1 interface lớn, khoảng 100 methods. Việc implements sẽ khá cực khổ, ngoài ra còn có thể dư thừa vì 1 class không cần dùng hết 100 method. Khi tách interface ra thành nhiều interface nhỏ, gồm các method liên quan tới nhau, việc implement + quản lý sẽ dễ hơn.

5. Dependency inversion principle



Dependency Inversion Principle

Would you solder a lamp directly
to the electrical wiring in a wall?

Nguyên lý cuối cùng, tương ứng với chữ **D** trong SOLID. Nội dung nguyên lý:

1. Các module cấp cao không nên phụ thuộc vào các modules cấp thấp.

Cả 2 nên phụ thuộc vào abstraction.

2. Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại.

(Các class giao tiếp với nhau thông qua interface,

không phải thông qua implementation.)

Nguyên lý này khá lát léo, mình sẽ lấy ví dụ thực tế. Chúng ta đều biết 2 loại đèn: đèn tròn và đèn huỳnh quang. Chúng cùng có **đuôi tròn**, do đó ta có thể thay thế đèn tròn bằng đèn huỳnh quang cho nhau 1 cách dễ dàng.



Ở đây, interface chính là **đuôi tròn**, implementation là **bóng đèn tròn** và **bóng đèn huỳnh quang**. Ta có thể swap dễ dàng giữa 2 loại bóng vì ổ điện chỉ quan tâm tới interface (đuôi tròn), không quan tâm tới implementation.

Trong code cũng vậy, khi áp dụng Dependency Inverse, ta chỉ cần quan tâm tới interface. Để kết nối tới database, ta chỉ cần gọi hàm Get, Save ... của Interface IDataAccess. Khi thay database, ta chỉ cần thay implementation của interface này.

Mình nói khá kĩ về nguyên lí này vì nó khá quan trọng. Về sau mình sẽ viết 1 bài riêng về **Dependency Injection** (**Dependency Injection chỉ là 1 trong những pattern để hiện thực Dependency**

Inversion, Dependency Injection != Dependency Inversion nhé các bạn), cùng với 1 số Dependency Injection Framework cơ bản.

Bài viết khá dài, xin có lời khen với các bạn đã kiên nhẫn đọc hết. Ở những bài viết sau mình sẽ giải thích rõ hơn về từng nguyên lý SOLID này, cùng với code minh họa + cách áp dụng nguyên lí vào quá trình code.

Lượt dịch từ bản gốc: <http://blogs.msdn.com/b/cdndevs/archive/2009/07/15/the-solid-principles-explained-with-motivational-posters.aspx>

Cột mốc 1000 rep trên stackoverflow – Khoe khoang thành quả và chia sẻ kinh nghiệm

Posted on 31/03/2015 by Phạm Huy Hoàng

LƯU Ý: ĐÂY LÀ MỘT BÀI VIẾT KHÔNG MANG TÍNH CHẤT TECHNICAL, CHỈ BAO GỒM VÀI LỜI KHOE KHOANG VÀ CHIA SẺ. NẾU CÁC BẠN MUỐN BỎ SUNG KIẾN THỨC, CÓ THỂ BỎ QUA BÀI NÀY VÀ CHỌN ĐỌC MỘT SÓ BÀI VIẾT BỔ ÍCH KHÁC NHƯ: [Linq](#), [Mock Data](#), [Meo văt Visual Studio](#), [C# hay ho](#), ... NHÉ.

Phần 1 – Khoe khoang

Đầu tiên, xin khoe cái biểu tượng trên stackoverflow của mình, nó là cái mà stackoverflow gọi là flair. Các bạn nhìn kỹ nhé, là **1200 REP** đó (Sau này có thể sẽ tăng thêm).



Bên trái là avatar của mình (Khá đẹp trai, các bạn không cần khen ...). Bên phải là nick stackoverflow, cùng với số rep kiếm được, phía dưới là đồng huy hiệu (Tương tự achievement trong game ấy mà).

Có bạn sẽ chắc lưỡi: Chỉ là con số 1000 thôi mà, có gì ghê gớm đâu. Thật ra, trên stackoverflow, số rep sẽ tương ứng với quyền hạn bạn có được => đóng góp càng nhiều thì quyền hạn càng cao, ở mức rep 1000 thì mình có một số quyền: Tạo phòng chat, xem review,.... Trong đó cái quyền mình thích nhất là cho phép hiện profile mở rộng, khi hover chuột vào avatar của mình, 1 khung text chứa toàn bộ thông tin profile của mình sẽ được hiển thị.

The screenshot shows a user profile with the following details:

- Reputation:** 1,500
- Create tags:** 1,000 (Achievement)
- Add new tags to the site:** You've been around for a while; see vote counts
- Moderation:** Help decide what questions and answers float to the top or participate in suggesting new features.
- Established user:** Create gallery chat rooms
- Create chat rooms where only specific users may talk:**
- Access review queues:** Access first posts and late answers review queues
- Communication:** Communicate with fellow users in chat rooms, meta-discussion, and comments.
- View close votes:**
- View and cast close/reopen votes on your own questions:**

```
}
```

And then, change your code to something like this

```
var pics = db.AlbumPictures.Where(p => p.AlbumID == album.ID)
    .OrderBy(p => p.RankOrder)
    .Select(p=>
        { dynamic copy = p.ToDynamic();
          copy.Thumbnail = ThumbManager.GetThumb(p.ID); //You can add more dynamic
          return copy;
        })
    .ToList();

return Json(pics, JsonRequestBehavior.AllowGet);
```

share edit delete flag

answered Jan 15 at 7:56



Huy Hoang Pham

1,273 • 1 • 11

Vietnam

to dicodedao.com

I'm a graduated student from FPT University.

I'm working as a junior developer at ASWIG Solution. I enjoy helping other fellow developers.

Visit my programming blog (Vietnamese) at <http://dicodedao.com/>

First, thanks for the answer. If this is the most elegant code to do what I want, Microsoft needs to make it easier to do this kind of thing. I'm beginning to understand why people think node.js is cool. – [John Sheehan](#)

I think converting the object to dynamic object is the best way to solve your problem. If this is the right answer. – [Huy Hoang Pham](#) Jan 16 at 1:35

Sau khi review lại thì thấy lượng rep mình kiếm được trong tháng cũng khá cao, nằm trong top 2% của tháng. Mặc dù không là gì so với các thánh nhưng tính ra cũng đáng tự hào rồi đây nhỉ :)).

Huy Hoang Pham [less info](#)

[edit](#) [privileges](#) [preferences](#) [fair](#) [apps](#) [my logins](#) [meta user](#) [network profile](#)

	bio	website	toidicodedao.com	I'm a graduated student from FPT University.
	location	Vietnam		I'm working as a junior developer at ASWIG Solution. I enjoy helping other fellow developers.
	age	23		
	visits	member for	4 months	Visit my programming blog (Vietnamese) at http://toidicodedao.com/
		visited	107 days, 4 consecutive	Contact me as: huyhoang8a5@gmail.com
		seen	6 mins ago	
1,273 reputation	stats	profile views	29	
		recent names	1	
+ 1 + 11	private	email	huyhoang8a5@gmail.com	
		real name	Huy Hoang Pham	

[summary](#) [answers](#) [questions](#) [tags](#) [badges](#) [favorites](#) [bounties](#) [reputation](#) [all actions](#) [responses](#) [votes](#)

81 Answers [votes](#) [activity](#) [newest](#) **1,273 Reputation** [top 2% this month](#)

[7 Building a dynamic JSON response in MVC 5 - How to tack on properties to...](#) [+10 In Angularjs, how do I use ng-repeat, ng-model, and ng-click to dynamically...](#)

[3 Controller with scope variable not working](#)

Khoe khoang thế đủ rồi, xin stop ở đây trước khi mọi chuyện trở nên quá lố (Tính mình cũng khá là khiêm tốn ...). Tiếp theo xin đến phần chia sẻ.

Phần 2 – Chia sẻ

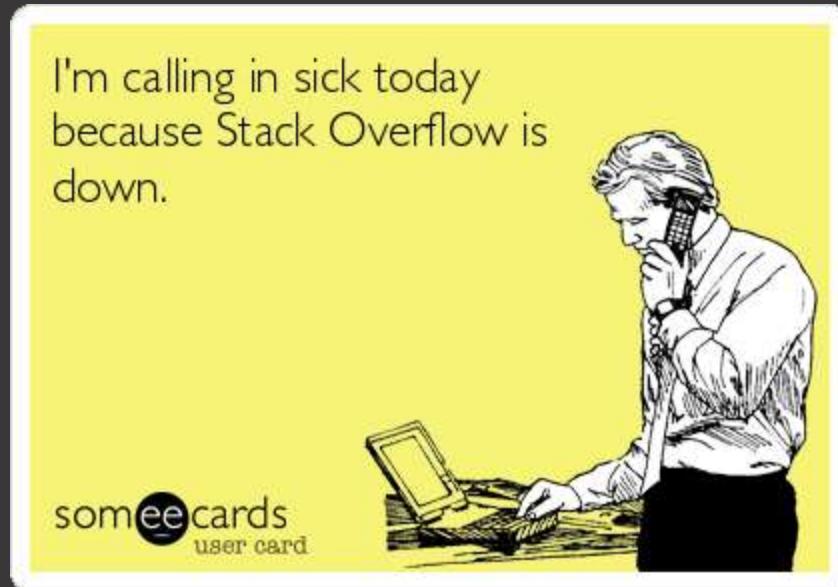
Đạt được mốc 1000 rep sau 4 tháng, vui cũng ít, tâm sự cũng nhiều, ở phần này mình chỉ muốn tóm gọn vài chuyện sau:

- Lý do mình tham gia “farm rep” trên stackoverflow.
- Chia sẻ kinh nghiệm cho các bạn muốn “farm rep”.

1. Lý do mình tham gia “farm rep” trên stackoverflow

Trang stackoverflow này thì quá nổi tiếng trong giới lập trình viên rồi (Gặp lỗi gì search google thì 90% đều dẫn tới trang này). Ban đầu, mình lâu lâu chỉ lên trang này search khi gặp exception, hoặc khi bí gì đó. Có lần, mình post câu hỏi nhưng không ma nào trả lời (Có vẻ do nó khó thật :v), quan sát kĩ thì thấy mấy thằng rep cao, huy

hiệu nhiều khi hỏi sẽ được trả lời nhiều hơn, tận tình hơn, cũng như được tôn trọng hơn (dĩ nhiên). Thời đó, do cũng khá rảnh nên mình cũng thử “farm rep” cho lên nhiều nhiều xem sao.



Phải công nhận, cơ chế rep + thứ hạng của stackoverflow khá là hay. Được càng nhiều rep, càng unlock được nhiều quyền lợi => muốn nhiều rep hơn, tương tự chơi game. Có lẽ nhờ thứ rep này mà stackoverflow mới trở thành cộng đồng Q&A lớn nhất. Tưởng chừng rep chỉ là thú vui tao nhã, chỉ là 1 con số để chung cho đẹp. Tuy nhiên, khi nghĩ lại, mình đạt được khá nhiều thứ sau khoảng 2 tháng “lăn lộn” trên stackoverflow:

- Do toàn phải đọc, trả lời câu hỏi bằng tiếng Anh nên khả năng reading và writing của mình tiến bộ lên trông thấy (Mình không muốn khoe là mình vừa có IELTS 7.5 đâu, bạn nào hỏi mình mới nói thôi).
- Có nhiều câu hỏi/trả lời khá thú vị. Tham gia quá trình hỏi, trả lời, khả năng technical mình cũng dần được nâng cao (Về C#, javascript, ASP.NET MVC, AngularJS). Nhiều câu trả lời còn phải tranh thủ trả lời nhanh/ trả lời kĩ vì lo “giang hồ” ks mất =))
- Sau này, khi cần thể hiện trình độ, hay đi phỏng vấn, chỉ cần đơn giản nói: Em có khoảng 1000 rep trên stackoverflow, nhiều câu trả lời với cái tag C#, ASP.NET, AngularJS, ... Đây sẽ là bằng chứng hùng hồn và thuyết phục về kinh nghiệm + khả năng giải quyết vấn đề của bạn.

81 Answers 1,273 Reputation

Answers

- 7 Building a dynamic JSON response in MVC 5 - How to tack on properties to...
- 3 Controller with scope variable not working
- 3 Efficiently paging large data sets with LINQ
- 2 Finding if single integer is in an array of integers
- 2 angular ui-grid event: row selected

[view more](#)

Questions

- 2 TagBuilder add attribute without value
- 0 Get only the changed paths of two revisions in SVN
- 0 Reuse AngularJS controller logic
- 0 Data isn't updated after called .CurrentValues.SetValues

Tags

Tag	Count
c#	1022
asp.net-mvc	114
javascript	120
angularjs	125
json	15
entity-framework	17
jquery	18
linq	19
.net	22
node.js	22

2. Chia sẻ kinh nghiệm cho các bạn muốn “farm rep”

Vì các lý do đã chia sẻ ở trên, mình hoàn toàn khuyến khích các bạn developer tham gia cộng đồng stackoverflow (Tốt cho cả bản thân và cộng đồng). Nhên đây, mình xin chia sẻ một số kinh nghiệm mình có được:

Ở phần trang chủ, mục favorite tag, hãy đánh vào một số tag mà bạn có kiến thức + hiểu biết về nó. Trường hợp của mình là C#, ASP.NET MVC. Stackoverflow sẽ lọc ra những câu hỏi mà bạn có thể trả lời, dựa theo những tag này.

Favorite Tags

- c#
- asp.net-mvc
- net
- javascript
- jquery
- angularjs
- linq
- sql
- sql-server
- twitter-bootstrap

Questions

- How do I know number of folders in the directory that is specified by user using FolderBrowserDialog in C#? (asked 1 min ago)
- Formulating Query (asked 1 min ago)
- Canvas - How to make line pass through a point (modified 1 min ago)

- Hãy rà soát bảng câu hỏi của stackoverflow khoảng 30p/1 tiếng 1 lần. Nên tập trung vào những câu hỏi chưa có người trả lời (Đôi khi có nhiều câu hỏi hay/lạ, mình cũng bấm vào xem để xem cách giải quyết thế nào).
- Lâu lâu, có những thằng các ké dạng hit-and-run (10-20 rep) hỏi câu hỏi, không miêu tả rõ ràng hoặc hỏi rất ngu. Nếu kiên nhẫn bạn có thể comment để hỏi rõ, còn không thì cứ bơ nó đi. Nhiều thằng tiếng Anh ngu cực, đọc câu hỏi không hiểu gì.
- Câu trả lời nên rõ ràng, ngắn gọn. Hãy tập format code trong stackoverflow (bấm dấu cách 4 lần), cách dùng [jsfiddle](#) để demo câu trả lời. Hãy nhớ: 1 câu trả lời được accept là 15 rep, được upvote là 10 rep. Nếu bạn trả lời tốt, có thể không được accept nhưng được upvote nhiều, cũng không tính là lỗ vốn.
- Một sự thực trớ trêu trong stackoverflow là: Có những câu quá đơn giản, hiển nhiên thì 30 giây vừa post đã có VÀI ĐÚA trả lời. Mình khá bức mình khi vừa viết câu trả lời xong đã có 1,2 thằng ks >"<, nhất là lũ Ân Độ. Có những câu phức tạp thì 3,4 tiếng chả thấy ai. Bạn nên thử sức mình với những câu hỏi này, có thể đó là những câu hỏi hay, cần suy nghĩ nhiều. Nếu đã trả lời đúng, người ta comment ok và quên accept thì nên nhắc khéo học accept câu trả lời, gỡ lại 15 rep.
- Điều cuối cùng: Vì mục đích tham gia là học hỏi, học hỏi và học hỏi, do đó các bạn đừng nên cay cú hay chửi thề hay blah blah gì, làm xấu mặt dân VN. Nhiều lúc mình trả lời xong thằng hỏi cao chạy xa bay mất, làm mình muối đào mồ má 4 đơi nhà nó ra. Các bạn hãy nhớ lâm nhảm câu khẩu quyết: Bình tĩnh tự tin, không cay cú.

Khoe khoang thì đã xong, chia sẻ thì cũng đã hết rồi. Lời cuối mình muốn nói là: chúc các bạn may mắn khi gia nhập stackoverflow nhé.

Viết unit test cho javascript với Jasmine

Posted on 02/04/2015 by Phạm Huy Hoàng

Blog có khá nhiều bài về code rồi nên hôm nay mình sẽ viết một bài để đổi gió.

1. Nhắc lại sơ về Unit Test

Trước khi có unit test, các lập trình viên thường code theo kiểu: code – test – fix lại – code tiếp – test lại – fix tiếp. Đôi khi chỉ vì sửa 1 lỗi nhỏ mà ta phải test lại rất nhiều lần. Để giải quyết vấn đề này, unit test và automation test ra đời. Mình không phải QA chuyên nghiệp nên không dám múa rìu qua mắt thợ, chỉ nói sơ về định nghĩa của 2 loại test này:

- **Unit test:** Đây là test **do developer** viết, được chạy để kiểm tra các hàm do developer viết ra có sai hay ko. Unit test thường được chạy mỗi khi build để đảm bảo các hàm đều chạy đúng sau khi ta sửa code.
- **Automation test:** Đây là test **do QA** viết, được chạy để kiểm thử hệ thống (Nếu không có automation test thì QA kiểm thử bằng tay, gọi là manual test).



Bài viết này chỉ tập trung vào unit test. Đây là test mà developer chúng ta phải viết. Các thư viện thường dùng để viết unit test là jUnit (Java), nUnit (C#), thư viện test của Microsoft, ... Một số khái niệm cần biết trong unit test:

- **Test case:** Mỗi function sẽ có nhiều test case, ứng với mỗi trường hợp function chạy. Những function đơn giản cần 2,3 test case, những function phức tạp thì cần > 10.

- **Setup:** Đây là hàm được chạy trước khi chạy các test case, thường dùng để khai báo dữ liệu, cài đặt các biến.
- **Teardown:** Đây là hàm được chạy sau khi các test case chạy xong, thường dùng để xóa dữ liệu, giải phóng bộ nhớ.
- **Assert:** Mỗi test case sẽ có 1 hoặc nhiều câu lệnh Assert, để kiểm tra tính đúng đắn của hàm. Vd: Assert.Equals(4, Add(2,2));
- **Mock:** Giả sử chương trình của bạn được chia làm 2 module: A và B. Module A đã code xong, B thì chưa. Để test module A, ta dùng **mock để làm giả** module B, không cần phải đợi tới khi module B code xong mới test được.



2. Giới thiệu về Jasmine

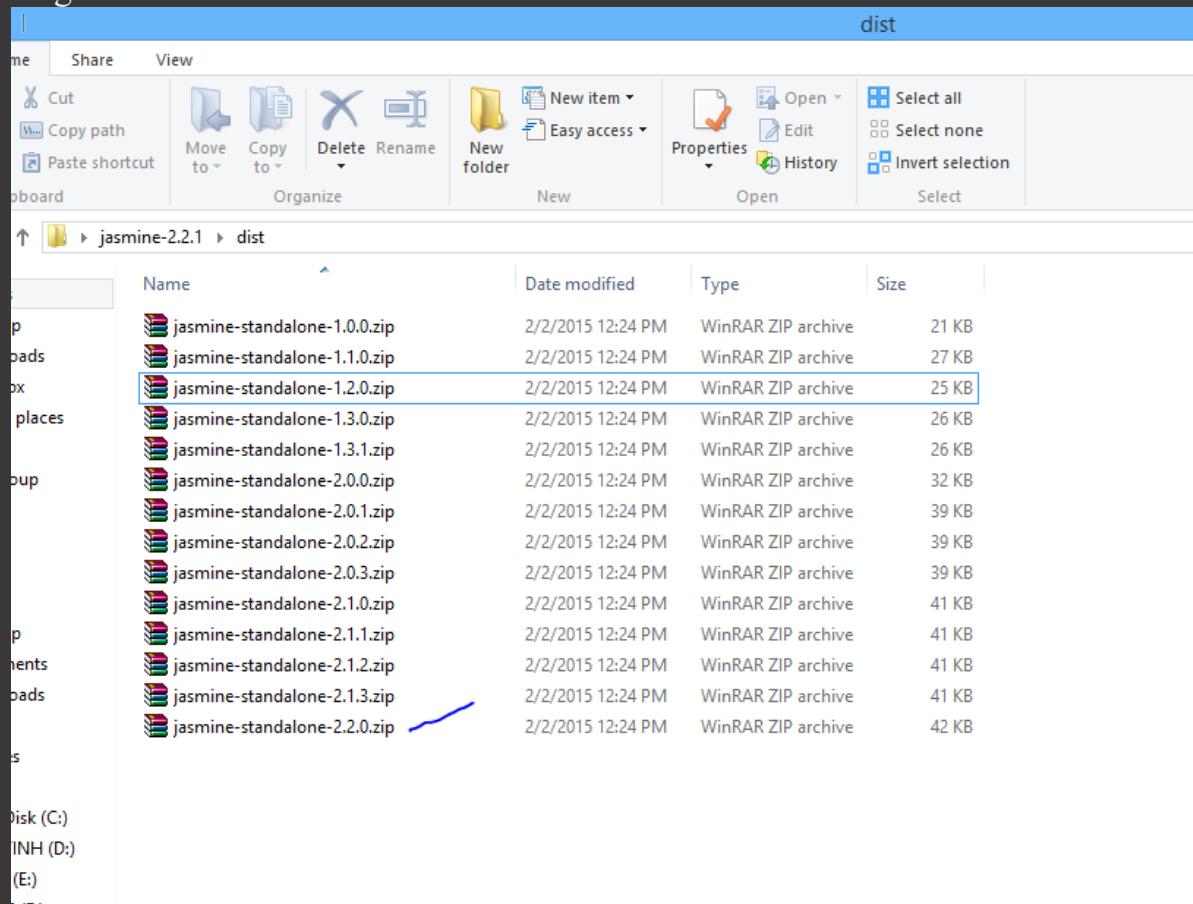
Ngày xưa ngay xưa, khi mà javascript là một ngôn ngữ trời đánh (xem thêm ở [đây](#)). Lúc đó javascript chưa có thư viện nào để viết unit test, cũng chưa có jQuery, do đó lập trình viên javascript cực kỳ khác con cò hó là mấy. Đến gần đây, javascript đã thành 1 ngôn ngữ hot, với vô số thư viện hỗ trợ. Jasmine là 1 trong 3 thư viện hỗ trợ

unit test mạnh nhất cho javascript hiện tại.

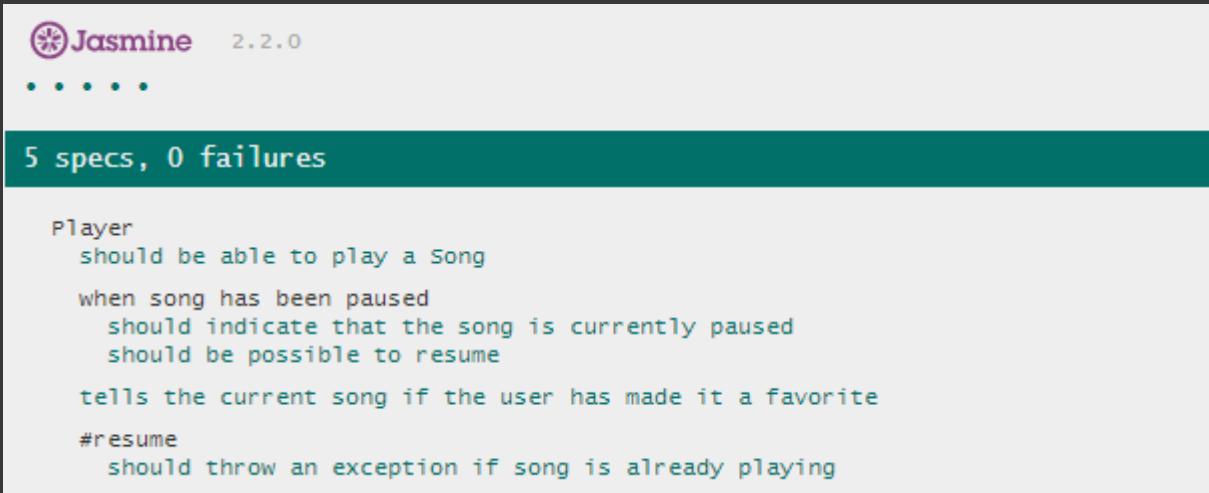


3. Demo Jasmine

Chúng ta bắt đầu làm demo nào, đầu tiên, các bạn tải jasmine bản mới nhất về ở đây: <https://github.com/jasmine/jasmine/releases> Giải nén, vào thư mục dist và giải nén bản mới nhất nhé:



Sau khi giải nén, ta được 1 thư mục mới, chạy thử file SpecRunner.html, ta có:



```

Jasmine 2.2.0
• • • •

5 specs, 0 failures

Player
  should be able to play a Song
  when song has been paused
    should indicate that the song is currently paused
    should be possible to resume
  tells the current song if the user has made it a favorite
  #resume
    should throw an exception if song is already playing

```

Xem source file html này, ta thấy gồm những file thư viện, file code và spec. Những unit test được viết trong file PlayerSpec.js:

```

1 <html>
2 <head>
3   <meta charset="utf-8">
4   <title>Jasmine Spec Runner v2.2.0</title>
5   <link rel="shortcut icon" type="image/png" href="lib/jasmine-
6 2.2.0/jasmine_favicon.png">
7   <link rel="stylesheet" href="lib/jasmine-2.2.0/jasmine.css">
8
9   <script src="lib/jasmine-2.2.0/jasmine.js"></script>
10  <script src="lib/jasmine-2.2.0/jasmine-html.js"></script>
11  <script src="lib/jasmine-2.2.0/boot.js"></script>
12
13  <!-- include source files here... -->
14  <script src="src/Player.js"></script>
15  <script src="src/Song.js"></script>
16
17  <!-- include spec files here... -->
18  <script src="spec/SpecHelper.js"></script>
19  <script src="spec/PlayerSpec.js"></script>
20 </head>
21 <body>
22 </body>
</html>

```

Để đơn giản, chúng ta sẽ tạo 1 file spec mới làm ví dụ, đặt tên là MySpec.js. Ta sửa lại file html như sau:

```

1 <html>
2 <head>
3   <meta charset="utf-8">
4   <title>Jasmine Spec Runner v2.2.0</title>

```

```

5
6  <link rel="shortcut icon" type="image/png" href="lib/jasmine-
7  2.2.0/jasmine_favicon.png">
8  <link rel="stylesheet" href="lib/jasmine-2.2.0/jasmine.css">
9
10 <script src="lib/jasmine-2.2.0/jasmine.js"></script>
11 <script src="lib/jasmine-2.2.0/jasmine-html.js"></script>
12 <script src="lib/jasmine-2.2.0/boot.js"></script>
13
14 <!-- include spec files here... -->
15 <script src="spec/MySpec.js"></script>
16
17 </head>
18
19 <body>
20 </body>
</html>

```

Trong file spec mới, ta sẽ viết 1 class Calculator, chuyên thực hiện các phép toán cộng trừ nhân chia, sau đó viết test case cho nó (Viết trong cùng file MySpec.js nhé) . Class Calculator

```

1 function Calculator()
2 {
3     this.add = function(a, b) { return a+b; };
4     this.minus = function(a, b) { return a-b; };
5     this.multiply = function(a, b) { return a*b; };
6     this.divide = function(a,b) {return a/b; } ;
7 }

```

Một số unit test cộng trừ nhân chia

- Hàm **describe** dùng để gom nhóm, ghi chú cho nhiều unit test.
- Hàm **it** tương đương với 1 unit test.
- Hàm **expect** chính là hàm assert để kiểm tra tính đúng đắn của unit test

```

1 describe("Cộng trừ", function() {
2     var cal = new Calculator();
3
4     it("Một với một là hai", function() {
5         expect(2).toBe(cal.add(1,1));
6     });
7
8     it("Hai với hai là bốn", function() {
9         expect(4).toBe(cal.add(2,2));
10    });
11

```

```

12 it("Năm trừ hai bằng ba", function() {
13   expect(3).toBe(cal.minus(5,2));
14 });
15
16 });
17
18 describe("Nhân chia", function() {
19   var cal = newCalculator();
20
21   it("Năm nhân hai bằng mươi", function() {
22     expect(10).toBe(cal.multiply(5,2));
23   });
24
25   it("Sáu chia hai bằng ba", function() {
26     expect(3).toBe(cal.divide(6,2));
27   });
28 });

```

Chạy lại file SpecRunner.html, ta được kết quả.

Jasmine 2.2.0

• • • • •

5 specs, 0 failures

Cộng trừ nhân chia

- Một với một là hai
- Hai với hai là bốn
- Năm trừ hai bằng ba

Nhân chia

- Năm nhân hai bằng mươi
- Sáu chia hai bằng ba

Giả sử ta cố ý sửa lại để hàm minus chạy sai, sẽ thấy unit test báo hàm sai như sau:

Jasmine 2.2.0

• • X • •

5 specs, 1 failure

[Spec List | Failures](#)

Cộng trừ nhân chia

- Một với một là hai
- Hai với hai là bốn
- Năm trừ hai bằng ba

Nhân chia

- Năm nhân hai bằng mươi
- Sáu chia hai bằng ba

Jasmine 2.2.0

• • X • •

5 specs, 1 failure

Spec List | Failures

Cộng trừ nhẫn chia Năm trừ hai bằng ba

```
Expected 3 to be 10.  
Error: Expected 3 to be 10.  
at stack (file:///C:/Users/HuyK20Hoang/Source/Repos/jasmine-sample/lib/jasmine-2.2.0/jasmine.js:1455:17)  
at buildExpectationResult (file:///C:/Users/HuyK20Hoang/Source/Repos/jasmine-sample/lib/jasmine-2.2.0/jasmine.js:1425:14)  
at Spec.Env.expectationResultFactory (file:///C:/Users/HuyK20Hoang/Source/Repos/jasmine-sample/lib/jasmine-2.2.0/jasmine.js:572:18)  
at Spec.addExpectationResult (file:///C:/Users/HuyK20Hoang/Source/Repos/jasmine-sample/lib/jasmine-2.2.0/jasmine.js:321:34)  
at Expectation.addExpectationResult (file:///C:/Users/HuyK20Hoang/Source/Repos/jasmine-sample/lib/jasmine-2.2.0/jasmine.js:516:21)  
at Expectation.toBe (file:///C:/Users/HuyK20Hoang/Source/Repos/jasmine-sample/lib/jasmine-2.2.0/jasmine.js:1379:12)  
at Object.<anonymous> (file:///C:/Users/HuyK20Hoang/Source/Repos/jasmine-sample/spec/MySpec.js:21:15)
```

Bài viết tới đây là kết thúc. Ở [phần 2](#), mình sẽ hướng dẫn chi tiết hơn về cách dùng hàm setup, teardown và mock với Jasmine. Các bạn có thể tải code sample của bài viết này ở đây: <https://github.com/conanak99/jasmine-sample>.

Viết unit test cho javascript với Jasmine – Phần 2

Posted on 07/04/2015 by Phạm Huy Hoàng

Tiếp nối [phần 1](#), ở phần này mình sẽ giới thiệu một số chức năng nâng cao của Jasmine, giúp việc viết unit test được dễ dàng hơn.

Nếu chưa tải Jasmine về máy, các bạn nên đọc lại phần 1 để biết chỗ tải về và cách viết 1 số test case cơ bản. Nội dung bài viết lần này bao gồm:

1. Một số matcher của Jasmine
2. Cách dùng các hàm before, after
3. Sử dụng spy và mock

1. Một số matcher của Jasmine

Matcher ở đây là 1 số hàm dùng để so sánh kết quả ta mong chờ (expected value) và kết quả trả về từ hàm (return value). Trong các ví dụ ở bài trước, mình đã dùng matcher **toBe**, matcher đơn giản nhất để viết test case.

```
1 it("Một với một là hai", function() {
2     expect(2).toBe(cal.add(1,1));
3 });
4
5 //Thêm not và trước matcher
6 it("Một với một không phải là ba", function() {
7     expect(3).not.toBe(cal.add(1,1));
8 });


```

Ta còn có thể so sánh 2 object, sử dụng matcher **toEqual**:

```
1 it("So sánh 2 object", function() {
2     var foo = {
3         a: 12,
4         b: 34
5     };
6     var bar = {
7         a: 12,
8         b: 34
9     };
10    //foo == bar có kết quả false
11    //Nhưng match toEqual vẫn so sánh đúng
12    expect(foo).toEqual(bar);
13 });


```

Một số matcher tiện lợi khác là: toBeDefined, toBeUndefined, toBeNull

```

1 it("Sử dụng toBeDefined", function() {
2     var a = {
3         foo: "foo"
4     };
5
6     expect(a.foo).toBeDefined();
7     expect(a.bar).not.toBeDefined();
8 });
9
10 it("Sử dụng toBeUndefined", function() {
11     var a = {
12         foo: "foo"
13     };
14
15     expect(a.foo).not.toBeUndefined();
16     expect(a.bar).toBeUndefined();
17 });
18
19 it("Sử dụng toBeNull", function() {
20     var a = null;
21     var foo = "foo";
22
23     expect(null).toBeNull();
24     expect(a).toBeNull();
25     expect(foo).not.toBeNull();
26 });

```

2. Cách dùng các hàm before, after

Jasmine hỗ trợ 4 hàm before, after như sau

```

1 describe("Sử dụng before và after", function() {
2
3     beforeEach(function() {
4         //Hàm này được chạy NHIỀU LẦN vào đầu mỗi test case
5     });
6
7     afterEach(function() {
8         //Hàm này được chạy NHIỀU LẦN vào cuối mỗi test case
9     });
10
11    beforeAll(function() {
12        //Hàm này được chạy MỘT LẦN duy nhất trước khi chạy
13        //các test case trong describe này
14    });
15
16    afterAll(function() {
17        //Hàm này được chạy MỘT LẦN duy nhất sau khi chạy

```

```
18 //xong các test case trong describe này
19 });
20
21 );
```

3. Sử dụng spy và mock

Trong Jasmine, spy được dùng để kiểm tra xem 1 hàm có được gọi hay không. Ngoài ra, nó còn có thể được dùng để làm mock object (Sẽ giải thích kĩ ở dưới).

Ở ví dụ này, ta sử dụng spy để kiểm tra xem hàm eat đã được gọi hay chưa:

```
1 describe("Dùng spy", function() {
2   var person, eaten = null;
3
4   //Hàm này được chạy đầu mỗi test case
5   beforeEach(function() {
6     person = {
7       eat: function(value) {
8         eaten = value;
9       }
10    );
11
12   //Gắn spy vào để theo dõi hàm eat của object person
13  spyOn(person, 'eat');
14
15   person.eat('banana');
16 });
17
18 it("tracks that the spy was called", function() {
19   expect(person.eat).toHaveBeenCalled();
20 });
21
22 it("tracks all the arguments of its calls", function() {
23   expect(person.eat).toHaveBeenCalledWith('banana');
24 });
25
26 it("stops all execution on a function", function() {
27   //Spy sẽ chặn hàm eat, do đó giá trị eaten không được set
28   expect(eaten).toBeNull();
29 });
30});
```

Khi cài đặt spy bằng hàm spy on, hàm được gọi tới đã bị chặn. Để hàm không bị chặn, ta setup thêm callThrough sau câu spy on

```

1 describe("Dùng spy", function() {
2   var person, eaten = null;
3
4   //Hàm này được chạy đầu mỗi test case
5   beforeEach(function() {
6     person = {
7       eat: function(value) {
8         eaten = value;
9       }
10    };
11
12   //Gắn spy vào để theo dõi hàm eat của object person
13   // Ở đây dùng thêm callThrough
14  spyOn(person, 'eat').and.callThrough();
15
16   person.eat('banana');
17 });
18
19 it("Not stop execution on a function", function() {
20   //Spy không chặn hàm eat nữa
21   expect(eaten).toBe('banana');
22 });
23 });

```

Trong trường hợp hàm eat chưa được viết xong thì sao? Trong quá trình code, điều này vẫn thường xảy ra. VD như module A phụ thuộc vào module B, nhưng module B chưa được viết xong. Để giải quyết trường hợp này, ta tạo 1 **module B giả**, tạm hoạt động như module B để cho module A gọi. Module B giả này được gọi là **mock**. Giả sử đối tượng person hàm foodEaten, nhưng viết chưa xong. Ta có thể dùng spy để làm mock, trả về giá trị cho hàm này

```

1 describe("Dùng spy", function() {
2   var person, eaten = null;
3
4   //Hàm này được chạy đầu mỗi test case
5   beforeEach(function() {
6     person = {
7       eat: function(value) {
8         eaten = value;
9       },
10      foodEaten: function() {
11        //Viết chưa xong
12      }
13    };
14
15   //Spy sẽ làm mock

```

```
16 //Giả kết quả trả về của hàm foodEaten
17     spyOn(person, 'foodEaten').and.callFake(function() {
18         return 'banana';
19     });
20 });
21
22 it("stops all execution on a function", function() {
23     //Gọi kết quả lấy từ hàm mock
24     expect(person.foodEaten()).toBe('banana');
25 });
26 }) ;
```

Trong phạm vi bài viết mình không thể giới thiệu hết 1 số tính năng khác của Jasmine, các bạn chịu khó tự tìm hiểu nhé.

Áp dụng LINQ trong javascript, chuyện nhiều người chưa biết

Posted on 14/04/2015 by Phạm Huy Hoàng

Nhu mình đã nói trong loạt bài: [Học ngôn ngữ lập trình nào bây giờ?](#), hiện tại Javascript đang trở thành 1 trào lưu mới. Gần đây, do đang tự học [Node.js](#) nên mình tập trung nghiên cứu javascript nhiều hơn. Vì vậy, trong khoảng thời gian này mình sẽ đăng nhiều bài viết liên quan đến javascript hơn, mong các bạn theo dõi.

Nhu ở bài viết về [LINQ](#), các bạn đã thấy sự mạnh mẽ và tiện dụng của LINQ trong C# (Java 8 chỉ mới cập nhật Stream API, mà vẫn còn thua LINQ nhiều lắm...). Để sử dụng 1 số hàm **tương-tự-LINQ** trong javascript, người ta thường dùng 1 số thư viện như:[underscore](#), [lodash](#), ... Trong bài viết này, mình sẽ giới thiệu 1 số function (**ít người biết**) sẵn có trong prototype Array của javascript, cũng có tác dụng tương tự như LINQ.

JavaScript vs. C# vs. other languages

JavaScript	C#
Dynamic	Static
Interpreted	Compiled
Client side	Serverside
No plugins required to run it	Need plugins to run in client like silverlight

Để dễ so sánh, với mỗi function trong java, mình sẽ đưa ra function tương tự trong C#. Mình sẽ dùng list dưới đây để làm ví dụ minh họa cho các function

```
1 var students = new List<Student>()
2 {
3     new Student() {Name = "Hoang", Age = 15},
4     new Student() {Name= "Minh", Age= 5},
```

```

5 new Student() {Name= "Sang", Age= 50},
6 new Student() {Name= "Long", Age= 27},
7 new Student() {Name= "Khoa", Age= 10},
8 };
1 var students=[
2     {Name: 'Hoang', Age: 20},
3     {Name: 'Minh', Age: 5},
4     {Name: 'Sang', Age: 50},
5     {Name: 'Long', Age: 27},
6     {Name: 'Khoa', Age: 10},
7 ];

```

1. filter

Trường hợp muốn lọc một số phần tử trong list, ta thường dùng hàm Where trong linq. Trong javascript, ta dùng hàm filter. Hàm này cần truyền vào 1 function để callback, với 3 tham số là: element, index, array.

```

1 var ageLess20 =
  students.Where(st => st.Age <
20);
1 var ageLess20 = students.filter(function(element,
2 index, array) {
3     return element.Age > 22;
4 });
5
6 //Nếu không quan tâm tới index hay array, ta có thể
7 viết
8 var ageLess20 = students.filter(function(st) { return
9 st.Age > 22; });
10
//Ở Firefox, hiện tại javascript đã lên phiên bản ES6
//Ta có thể dùng arrow function, tương tự lambda
expression
var ageLess20 = students.filter(st => st.Age > 22);

```

2. every và some

Trong LINQ, ta thường dùng hàm All để kiểm tra **toàn bộ phần tử thỏa mãn 1 điều kiện**, hàm Any để kiểm tra nếu **ít nhất 1 phần tử thỏa mãn 1 điều kiện**. Trong javascript, 2 hàm ta sử dụng là every và some

```

1 //Kiểm tra toàn bộ students đều lớn hơn 5 tuổi
2 bool allStudentLarger5 = students.All(st => st.Age > 5); //
3 false
  bool anyStudentAge50 = students.Any(st => st.Age == 50); // true
1 var allStudentLarger5 = students.every(function(st) { return
2 st.Age > 5 }); // false

```

```
1 var anyStudentAge50 = students.some(function(st) { return st.Age == 50 }); // true
```

3. forEach

Hàm này công dụng cũng như, sẽ duyệt qua 1 lần tất cả các phần tử của list, và gọi function được truyền vào.

```
1 //Duyệt từng học sinh trong list, in ra
2 students.ForEach(Console.WriteLine);
1 //Với javascript
2 students.forEach(console.log);
```

4. map

Trong LINQ, đôi khi ta cần lọc ra 1 số trường của các object trong list, ta thường sử dụng Select. Trong javascript, hàm tương tự được sử dụng là hàm map

```
1 //Lọc lấy 1 list toàn tên của học sinh
2 var names = students.Select(st => st.Name);
3 //["Hoang", "Minh", "Sang", "Long", "Khoa"]
1 var names = students.map(function(st){ return st.Name });
2 //['Hoang', 'Minh', 'Sang', 'Long', 'Khoa']
```

5. reduce

Đây là hàm thay thế cho toán tử Aggregate trong LINQ. Toán tử này hơi khó giải thích, các bạn vui lòng đọc thêm ở

đây: <http://stackoverflow.com/questions/7105505/linq-aggregate-algorithm-explained>

6. find và findIndex (hiện tại chỉ firefox và safari)

Để tìm 1 phần tử nằm trong list, với LINQ ta thường dùng FirstOrDefault (Để lấy chính phần tử đó), hoặc FindIndex (Để tìm index của phần tử đó trong list). Lưu ý là 2 hàm này đang trong giai đoạn thử nghiệm, chỉ **chạy được trên firefox và safari** thôi nhé.

```
1 //Tìm học sinh có tuổi bằng 50
2 Student found = students.FirstOrDefault(st => st.Age == 50);
3 //Student Long, Age = 50
4
5 //Tìm index của học sinh đó
6 int index = students.findIndex(st => st.Age == 50); //2
1 //Chi chạy được trong firefox và safari
2 var found = students.find(function(st){ return st.Age = 50 });
3
4 var index = students.findIndex(function(st){ return st.Age = 50 });
```

Nếu bạn thấy những function này không đáp ứng được nhu cầu, hãy chờ ở những bài sau, mình sẽ giới thiệu lo-dash, một thư viện với vô số function hỗ trợ việc duyệt và xử lý mảng. Mình đảm bảo sau khi sử dụng quen, bạn sẽ không thể code javascript mà thiếu nó (Giống như 90% developer C# bây giờ không thể sống thiếu LINQ vậy).

Kết luận: Theo ý kiến cá nhân, Javascript y hệt **nhus con gái**. Nhìn bên ngoài, có vẻ nàng là một cô nàng khá hot, nhiều người theo đuổi (1 lô 1 lốc các framework viết cho javascript). Mới quen, lại thấy nàng khá rỗi rãm, phức tạp, đôi khi hơi điên điên, đôi lúc lại hơi dẽ dãi (Dẽ dãi là vì đôi khi code js sai bét nhè chả thấy nó nói câu nào, còn về sự điên điên của js thì các bạn nên google để tìm hiểu thêm: <http://webreflection.blogspot.com/2012/10/javascript-made-everyone-crazy.html>). Có điều, sau khi đã quen 1 thời gian lại thấy nàng rất dễ thương, tiện dụng, cái gì cũng biết. Tóm gọn lại, xin lấy tấm hình này để kết thúc bài viết.



Cái nhìn của tác giả về javascript: Xưa và nay.

Review sách: Clean Code: A Handbook of Agile Software Craftsmanship

Posted on 09/04/2015 by Phạm Huy Hoàng

Hôm nay bỗng dung không có hứng viết bài về technical, thôi thì lôi đại cuốn này ra review vây. Mình đọc cuốn này trong thời gian còn làm việc ở FPT Software (Làm việc lúc nào cũng dư thời gian nên toàn lôi ebook ra đọc. Cuốn sách này xứng đáng là **sách gói đầu giường của mọi developer**. Mình khuyên các bạn nên mua bản gốc, 1 là để đọc, 2 là nếu gặp thằng nào code ngu, có thể cầm cuốn này **đập vào đầu nó và bắt nó đọc**.



Giới thiệu

Như cái tên “Clean Code”, đây là cuốn sách hướng dẫn các bạn developer viết ra “code sạch”. Thế nào là code sạch? Theo định nghĩa của sách, đó là code dễ đọc, dễ hiểu, dễ sửa chữa và bảo trì. Có bạn sẽ xì mũi bảo: Giờ, có gì đâu, cái đấy ai code chả được. OK! Mời bạn thử xem lại suorce code của 1 project mình đã làm cách đây 3-6 tháng xem, có hiểu được mình viết gì ko? Nếu không tức la code của bạn chưa được sạch lắm đâu =))).

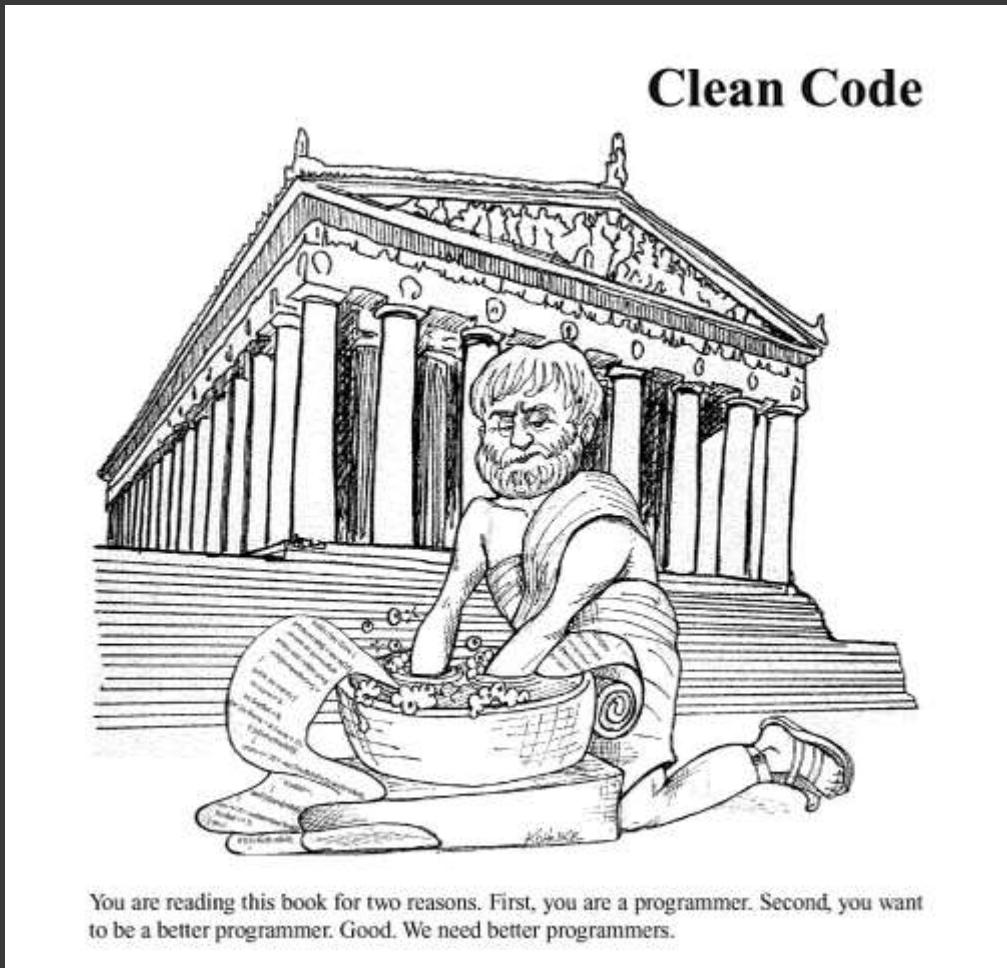
Đa số developer đều code “không được sạch” (cả mình ngày xưa vẫn thế), có nhiều nguyên nhân dẫn đến chuyện này. Từ nguyên nhân chủ quan như: nghĩ mình trình cao, không biết code, code đại ... cho tới khách quan như: bị leader dí deadline, code

cho xong, sau này rảnh optimize sau (“**Sau này**” đối với developer là “**không bao giờ**”). Đôi khi bạn vào 1 dự án cũ, thấy 1 **đống bầy nhầy**. Bạn sẽ phải vừa mò code vừa chửi tổ tông tám đời nhà cái thằng developer cũ vì nó code vừa ngu vừa khó hiểu. Đó là hậu quả của việc “code không sạch”.

Vì nãy giờ toàn khen nên cũng không muốn nhận xét gì nhiều, mình chỉ tóm tắt vài điều hay bạn có thể học được trong sách

Bài học rút ra

- Tâm quan trọng của việc viết “code sạch”.
- Cách đặt tên biến, tên hàm. Tên biến, tên hàm phải nói rõ tác dụng của hàm và biến. (Vâng, những thằng mình muốn chơi gạch là những thằng đặt tên biến int a, b, c hoặc đặt tên hàm tối nghĩa).
- Độ dài của hàm, các parameter truyền vào. (1 hàm đừng nên dài quá 1 trang A4, cũng đừng nên có quá nhiều parameter. Thủ đọc 1 hàm dài khoảng 800 dòng code các bạn sẽ biết cực chừng nào).
- Tại sao không nên lạm dụng comment. Comment không phải xấu, nhưng nhiều người viết code không sạch, sau đó dùng comment để nói đoạn code đó **làm gì**. Sách khuyên ta nên viết code tự comment, tức là đoạn code đã trong sáng tới mức đọc là biết **code làm gì**, comment chỉ nên dùng để bổ sung những điều không giải thích được qua code (VD: đoạn code này để fix bug abc, dùng thuật toán này vì lý do bcd).
- Hướng dẫn cách viết và dùng unit test.
- Giải quyết 1 số vấn đề liên quan tới concurrency.
- Một số ví dụ cụ thể về việc refactor code (Phần này khá hay, biến code rỗm thành code sạch thông qua các biện pháp refactor).
- Một số dấu hiệu nhận biết code smell (Phần này cũng khá hay, bạn có thể đọc và dựa theo các dấu hiệu này để tìm những đoạn “code rỗm” trong project hiện tại =))).



You are reading this book for two reasons. First, you are a programmer. Second, you want to be a better programmer. Good. We need better programmers.

Lời khuyên của mình: Ngay khi vừa ra trường hoặc đã code đc khoảng 2-3 tháng, bạn nên đọc cuốn này để tạo dựng những thói quen tốt cơ bản khi code. Sau khi code đc khoảng 1-2 năm, hãy đọc lại 1 lần nữa để nghiệm lại những điều mình chưa hiểu lần đầu đọc. Hai câu nói mình tâm đắc nhất trong sách sau khi đọc:

1. Code cho máy đọc thì ai cũng viết đc, code cho người đọc thì chỉ có developer giỏi mới viết đc.
2. Hãy code như thể thằng developer bảo trì code của bạn là 1 thằng sát nhân bệnh hoạn biết địa chỉ nhà của bạn. (Hiện giờ nhiều developer code cho xong function rồi để đó. Cứ tưởng tượng thằng developer bảo trì code của bạn mà đọc những dòng code bạn viết sẽ làm gì bạn sau khi đọc code bạn viết. Mình là mình cũng từng muốn cầm dao lùi mấy thằng developer cũ của team nhiều lần lắm đấy).

Lập trình viên “trình cao” thì nên đọc sách gì? – Phần 1

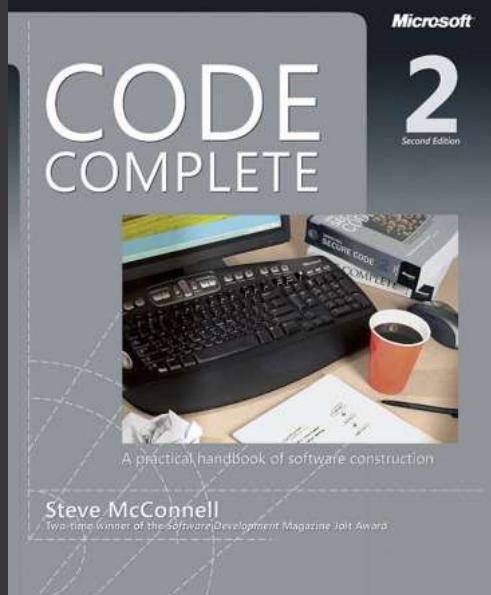
Posted on 21/04/2015 by Phạm Huy Hoàng

Đầu tiên, xin hứng chịu gạch đá từ nhiều bạn rằng: developer thì cần gì phải đọc sách, code nhiều là giỏi thôi. Vâng, các cậu có cu, nhầm, các cụ đã có câu là “practice make perfect”, cứ làm hoài là giỏi. Tuy nhiên, phải làm đúng cách thì mới giỏi được, code dở mà không chịu tìm cách cải thiện kĩ năng code, cứ code hoài 1 kiểu cũ thì bao giờ mới giỏi được.

Về sách lập trình mình đọc cũng được kha khá, sách hay có dở có. Tuy nhiên mỗi cuốn sách hay hay dở đều làm mình ngộ ra được vài điều. Khảo sát trong cuốn **Code Complete** cho thấy trung bình 1 developer đọc ít hơn 1 cuốn sách mỗi năm. Chỉ cần các bạn làm theo mình, mỗi năm đọc ít nhất một cuốn, các bạn sẽ giỏi hơn khoảng 90% developer còn lại rồi nhé.

Trong danh sách mình giới thiệu tiếp theo, có cuốn mình đã đọc, có cuốn mình đọc nửa chừng rồi bỏ, có cũng cuốn mình chưa đọc. Những cuốn đã đọc mình sẽ chia sẻ cảm nhận của mình, còn cuốn nào chưa đọc xin phép dịch nhận xét của những người khác vậy, các bạn thông cảm. Dưới đây là danh sách những cuốn sách developer nên đọc, được giới thiệu bởi [codinghorror](#), một blog IT khá nổi tiếng:

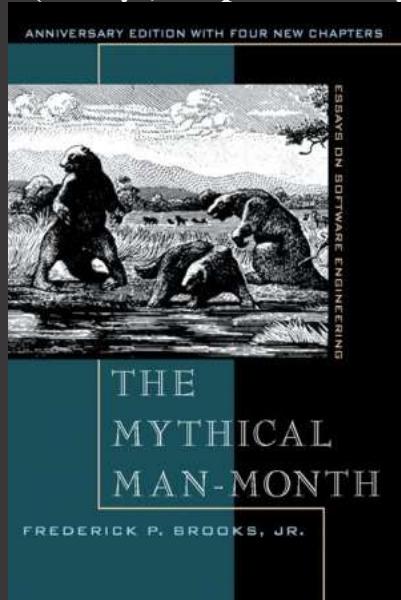
1. **Code Complete (Đã đọc, drop nửa chừng)**



Nếu bạn muốn theo đuổi công việc lập trình một cách nghiêm túc, bạn nên đọc cuốn này. Mình được một ông anh giới thiệu (mà chắc lão cũng chẳng đọc hết). Có thể nói là đọc tới đâu ngộ ra tới đấy. Trong quá trình code, có lúc bạn sẽ gặp các trường hợp như: tách method thế nào, chia class ra sao, đặt tên biến thế nào, ... Cuốn sách này sẽ là người thầy, người anh của bạn, với vô số hướng dẫn từ tổng quan như: xây dựng kiến trúc, liên hệ giữa các component, ... cho tới chi tiết như: cách tổ chức function, cách đọc tên biến.

Đọc tới giữa sách, do nó hơi sa đà vào C++, và lại con Kindle Fire của mình vừa bị hư nên mình đành drop, chưa có thời gian đọc lại.

2. The Mythical Man-month (Đã đọc, drop sau khi đọc xong 14/20 chương)



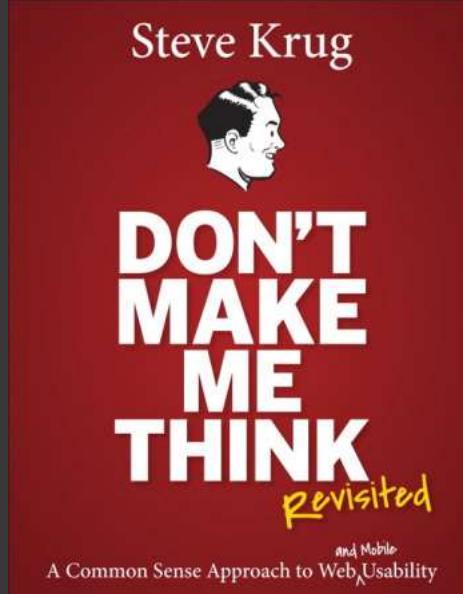
Đây là một cuốn sách về khùng long trong ngành IT? Đùa đấy! Nó là một cuốn sách khá nổi tiếng (Không thua Kinh Thánh) trong giới manager và PM ngành IT. Như lời cuốn sách đã nói: **Máy tính có thể thay đổi, nhưng con người thì không.** Bằng chứng là những điều cuốn sách nói cách đây 30 năm bây giờ vẫn còn đúng trong ngành phần mềm:

- Phép tính man-month: 1 project cần 3 người là trong 4 tháng tức $3 \times 4 = 12$ man month. Nếu tính theo cách bình thường thì 6 người sẽ cần $12/6 = 2$ tháng. Tuy nhiên, man-month là một thứ kì quái, thêm người vào dự án chưa chắc sẽ tăng tốc độ dự án (Thời mình làm bên FSoft lầm cụ ko biết điều này >.<).

- Vấn nạn document: Việc quản lý document rất mất thời gian và vô cùng phức tạp, có 1 số dự án document còn dài + rắc rối hơn cả code (Thời nay áp dụng Agile thì đỡ...).

Nói chung, cuốn sách này sẽ không giúp bạn code giỏi hơn, nhưng nó giúp bạn có cái nhìn tổng quan về những mặt khuyết thiêu của ngành phần mềm. Những kiến thức này sẽ rất có ích nếu bạn leo lên vị trí manager hoặc PM v...v. Khuyến cáo: Sách dùng ngôn từ hơi cổ, có đôi lúc giống tiểu thuyết, mình đọc còn xâm.

3. Don't make me think (Đã đọc hết)

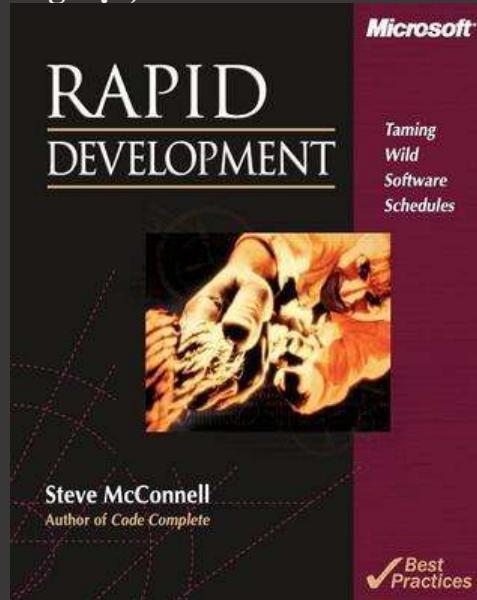


Một cuốn sách rất hay về thiết kế giao diện. Nó đưa ra một qui tắc rất đơn giản và hữu dụng trong thiết kế UI: Người dùng rất lười, hãy thiết kế sao cho người dùng ít suy nghĩ nhất. Cuốn sách không hướng dẫn cách thiết kế đẹp, nhưng hướng dẫn cách thiết kế đơn giản nhất, dễ sử dụng nhất, đỡ tốn công sức người dùng.

Sách còn hướng dẫn một số control nên dùng khi thiết kế web: form, checkbox, radio, dropdown, ... và cách sử dụng những control này hợp lý. Ngoài ra còn có 1 câu chuyện về “1 button đáng giá 500.000\$” trong sách, về sự đắt giá của thiết kế UI. Chỉ thêm 1 nút vào trang web có thể tăng doanh thu lên đến 500.000\$ trong 1 năm, các bạn có thể tìm hiểu đọc thử.

Quên, sau này thấy thằng nào thiết kế giao diện loằng ngoằng, rắc rối, nhiều bước khó sử dụng, hay cầm cuốn này đập vào mệt nó nhé.

4. Rapid development (Đang đọc)

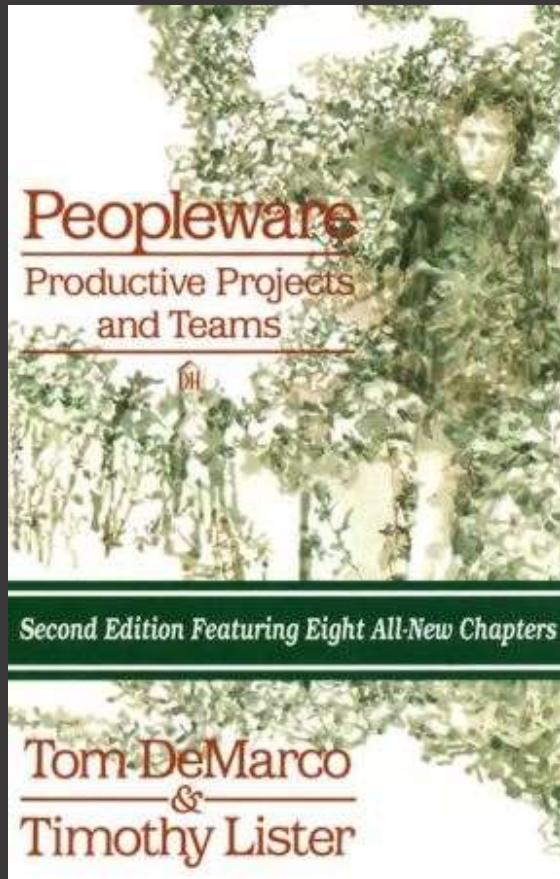


Ngày xưa ngay xưa, khi mà nhà nhà dùng waterfall, người người dùng waterfall, công việc của developer khá đơn giản, chỉ là code theo design. Thế rồi requirement ngày càng đổi xoành xoạch, các project waterfall chết hàng loạt, do đó người ta chuyển qua Agile.

Thế nhưng chuyển qua Agile, vẫn có vô số project IT bị chết. Cuốn sách này là một trong những cuốn sách đi đầu trong trào lưu Agile. Nó đưa ra những cách áp dụng Agile hợp lý, chỉ ra những sai lầm thường gặp trong quá trình sử dụng Agile như: Phình requirement, code xuống cấp, ... cũng như đưa ra 1 số case study để học hỏi.

Bản thân mình chưa đọc, nhưng sau khi xem giới thiệu có lẽ sẽ tái về đọc thử.

5. Peopleware (Đã đọc hết)



Cuốn này mình chợt thấy khi đang tìm ebook trên it-ebook. Thấy review trên amazon có vẻ cao nên xem thử.

Cuốn sách này không nói gì về code, mà tập trung vào yếu tố con người – yếu tố quan trọng nhất, trong quá trình phát triển phần mềm. Nó đáng là cuốn sách gói đầu giường của các team leader, PM, manager. Một số nội dung trong sách: Tạo môi trường làm việc thuận lợi cho developer, cách estimate dự án, cách xây dựng một team mạnh và vững chắc,

Vì mình vẫn còn là junior developer nên đọc cuốn này cũng không thấy hữu dụng lắm, đọc cho biết + giải trí thôi, coi như chuẩn bị kiến thức cho con đường manager sau này =))

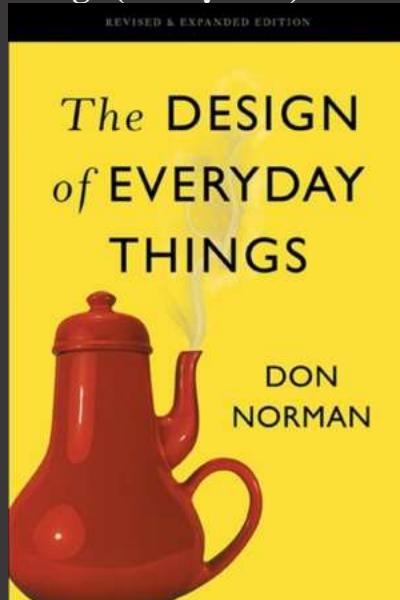
Bản gốc tham khảo: <http://blog.codinghorror.com/recommended-reading-for-developers/>

Lập trình viên “trình cao” thì nên đọc sách gì? – Phần 2

Posted on 23/04/2015 by Phạm Huy Hoàng

Nối tiếp [phần 1](#), ở phần này mình sẽ giới thiệu những cuốn sách còn lại trong danh sách được giới thiệu trên codinghorror. Có vài cuốn hơi cao siêu, các bạn nên đọc theo tính chất “giải trí, học hỏi”, nếu giữa chừng tẩu hỏa nhập ma có thể ngừng cũng được, không sao =)))

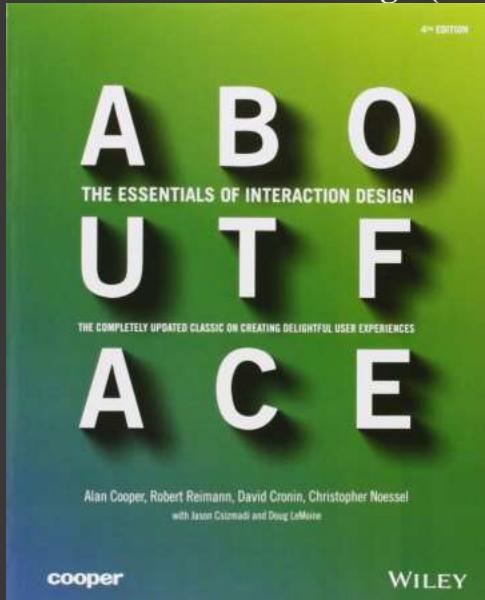
6. The Design of Everyday Things (Đã đọc hết)



Cuốn sách này được viết bởi [Don Norman](#). Ông là một bậc thầy về tâm lý học và design. Ông đưa ra khái niệm “user-centered design” (Design hướng về phía người dùng), **chú trọng vào tính hữu dụng và dễ sử dụng của thiết kế**, yếu tố thẩm mĩ chỉ là phụ.

Ý kiến cá nhân mình thì cuốn này khá hay, ảnh minh họa rất nhiều. Cuốn sách này sẽ làm thay đổi tư duy thiết kế của bạn, do đó dân design hay lập trình thì đều nên đọc. Như tựa đề, cuốn sách phân tích thiết kế của các vật dụng thường ngày: cánh cửa, tủ lạnh, máy lạnh, ... cho tới điện thoại, hệ thống máy tính, ... đồng thời phân tích tại sao design này thành công, design kia thất bại. Đảm bảo các bạn sẽ nhiều lần gật gù “à ra thế” khi đọc cuốn này.

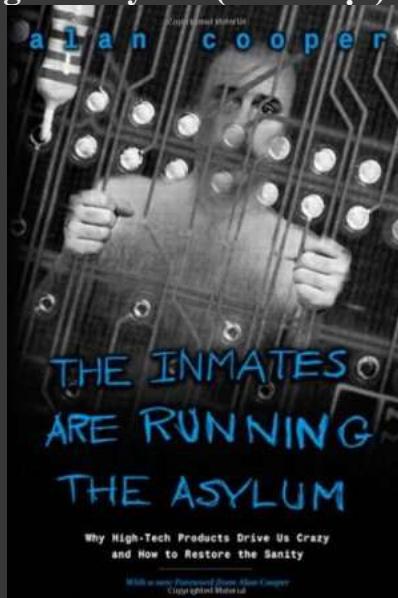
7. About Face: The Essentials of Interaction Design (Chưa đọc)



Tác giả cuốn này là Alan Cooper, **cha đẻ của Visual Basic**.

Mình chỉ xem sơ qua mục lục, đoạn đầu khá cao siêu, nói về phân tích mô hình tư tưởng của người dùng này nọ, mấy chương cuối lại về cách sử dụng các control, các yếu tố tạo thành 1 design tốt v...v. Thêm vào list cho đủ thôi, bạn nào đã đọc rồi thì cho mình xin thêm ý kiến

8. The Inmates Are Running the Asylum (Chưa đọc)



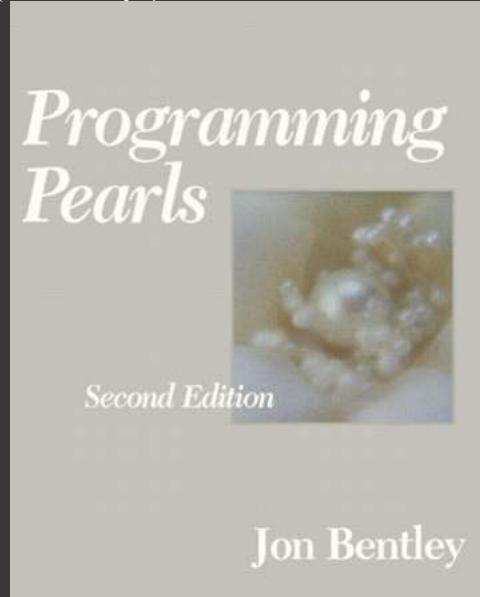
Nhìn cái bìa có vẻ ảo, mình mới nhìn cứ tưởng tiêu thuyết trinh thám kinh dị hay cái gì đại loại thế. Cái tựa tạm định là “Lũ tù phạm quản lý nhà thương điên”. Cuốn này

cùng tác giả với cuốn số 7 phía trên, cũng viết về user experience, trải nghiệm người dùng.

Đọc sơ được mục lục và 1,2 chương đầu thì mình thấy cuốn này khá có tiềm năng. Alan Turing đã chỉ ra mặt trái của công nghệ: Việc ta **thêm software vào một thiết bị sẽ làm thiết bị đó ngày càng phức tạp và khó sử dụng** (Thử so sánh máy ảnh, di động ngày xưa và máy ảnh digital, smartphone bây giờ).

Tiếp theo, sách giải thích tại sao 1 số hệ thống lại khó sử dụng (như máy ATM), lý do là: **Vì sử dụng như vậy sẽ tiện cho developer lập trình hơn**. Chắc các bạn developer chắc ai cũng có lúc như mình, implement một chức năng bằng cách dễ code + code nhanh nhất, đổi lại có thể bắt người dùng vất vả hơn một chút. Sau khi đọc xong cuốn này, có lẽ mình sẽ viết 1 bài review riêng cho nó.

9. Programming Pearls (Chưa đọc)

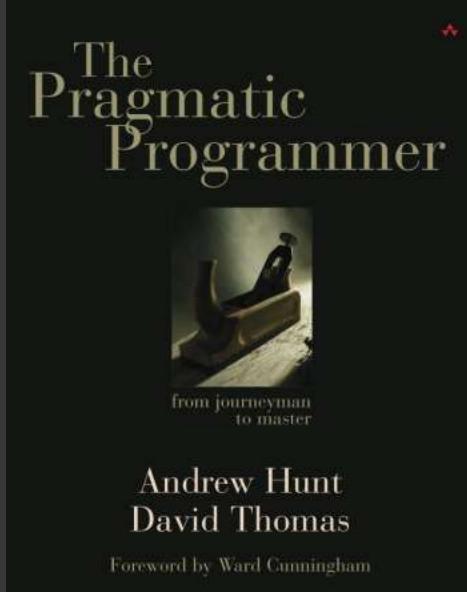


Đọc cái tựa, mình tưởng là sách dạy lập trình cho ngôn ngữ Pearl, hóa ra không phải. Nội dung sách đúng như tựa, nó là “**Những viên ngọc trong nghệ thuật lập trình**“. Nó là những viên ngọc được các lập trình viên kì cựu đúc kết qua bao thế hệ. Sách hơi nhấn mạnh tới các thuật toán này nọ, bạn hoàn toàn có thể skip (Giờ sort, hash v...v thì chỉ cần gọi 1 hàm từ API là xong). Những phần tinh túy trong sách này bạn nên đọc là:

- Cách lựa chọn thuật toán phù hợp, đưa ra performance tối đa.

- Cách estimate bằng thông, lưu lượng. Đây là thứ khá hay, ít sách viết. Bạn có thể sử dụng những kỹ năng trong sách để ước tính lưu lượng người dùng chương trình, dung lượng ô cứng cần thiết cho DB, rất hữu ích.

10. The Pragmatic Programmer: From Journeyman to Master (Đã đọc hết)



Tác giả của cuốn [The Passionate Programmer](#) đã lấy cảm hứng dựa trên cuốn sách này. Đây là một cuốn sách xuất sắc dành cho developer, nhưng không nói nhiều về code, mà nói về **cách code, cách tiếp cận và giải quyết vấn đề**

Nếu lười, các bạn có thể xem bản tóm tắt (chỉ 1 trang) ở đây:<http://blog.codinghorror.com/a-pragmatic-quick-reference/>. Mình nghĩ nó sẽ khá là có ích, không ít thì nhiều.

Danh sách tới đây là hết, bạn nào muốn bổ sung hay giới thiệu sách thì xin comment nhé.

Bản gốc tham khảo: <http://blog.codinghorror.com/recommended-reading-for-developers/>

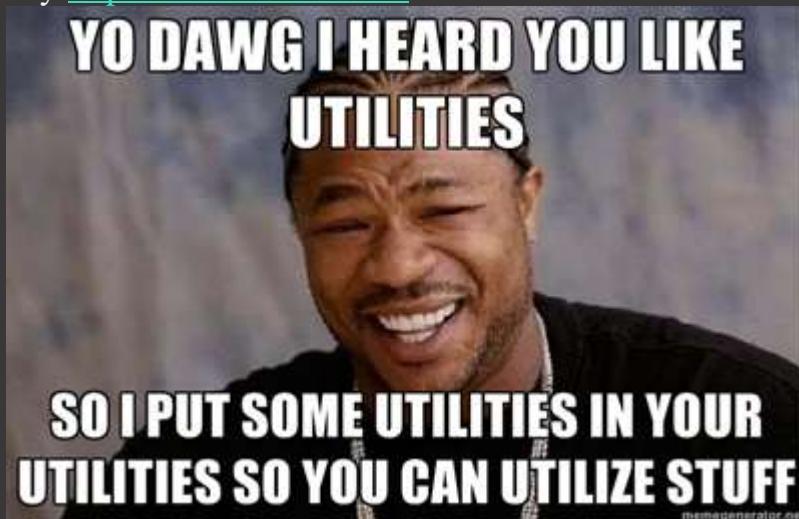
Tăng sức mạnh cho javascript với lodash

Posted on 16/04/2015 by Pham Huy Hoàng

Như đã nói ở bài trước, lần này mình sẽ giới thiệu 1 thư viện javascript vô cùng bá đạo có tên là “[lodash](#)”, có thể nói nó là LINQ trong javascript. Đảm bảo chỉ sau 1 lần dùng thử, thư viện này sẽ trở thành thư viện không thể thiếu trong mỗi project javascript của bạn.

1. Giới thiệu tổng quan về lodash

Tiền thân của lodash là [underscore](#) – một thư viện javascript cũng khá nổi tiếng (Bạn nào hỏi: Nổi tiếng sao mình ko biết?... vui long đi chỗ khác chơi nhé :)). Có thể xem lodash là 1 bản mở rộng, với nhiều chức năng hơn, performance cao hơn underscore. Lodash cung cấp rất nhiều chức năng, chia làm vài nhóm như: chức năng linh tinh (check null, underscore, ..), chức năng hỗ trợ xử lý string, chức năng xử lý object, chức năng xử lý array. Vì phạm vi bài viết có hạn, mình chỉ ví dụ và đưa ra một số chức năng chính, các bạn có thể thao khảo danh sách API full của lodash ở đây:<https://lodash.com/docs>



2. Demo – 1 số function của string và function linh tinh

Đây là trang fiddle mẫu mình đã include lodash sẵn, các bạn có thể vào và code theo các ví dụ mình cung cấp để chạy thử code

<http://jsbin.com/fijubokiqe/1/edit?js,console,output>

Các chức năng linh tinh của lodash:

`_.isUndefined`, `_.isNull`, `_.isFunction`, `_.isNumber`, `_.isObject`: Đây là một số function để kiểm tra kiểu của biến, cũng như xem biến có undefined, null hay ko?

```

1      _.isNull(null);
2      // → true
3
4      _.isFunction(_);
5      // → true
6      _.isFunction(/abc/);
7      // → false
8
9      _.isNumber(8.4);
10     // → true
11     _.isNumber(NaN);
12     // → true
13     _.isNumber('8.4');
14     // → false
15
16     _.isObject({ });
17     // → true
18     _.isObject([1, 2, 3]);
19     // → true
20     _.isObject(1);
21     // → false

```

_.isEqual: So sánh 2 array, hoặc 2 object. Function này sử dụng **deep comparision**, so sánh từng field của 2 object, hoặc từng phần tử của 2 array nên cho kết quả đúng như mong muốn nhé

```

1      var object = { 'user': 'fred' };
2      var other = { 'user': 'fred' };
3
4      object == other; // → false
5      _.isEqual(object, other); // → true

```

```
6
7     var array = ['hello', 'goodbye'];
8     var others = ['hello', 'goodbye'];
9
10    array == others; // → false
11    _.isEqual(array, others); // → true
```

Một số function của string:

_.camelCase: Biến string thành chuỗi camel case

_.capitalize: Viết hoa kí tự đầu của string

_.pad, _.padLeft, _.padRight: Canh giữa, trái, phải string bằng cách thêm khoảng trắng vào string

```
1     _.camelCase('Foo Bar');
2     // → 'fooBar'
3     _.camelCase('--foo-bar');
4     // → 'fooBar'
5
6     _.capitalize('fred');
7     // → 'Fred'
8
9     _.pad('abc', 8);
10    // → ' abc '
11    _.padLeft('abc', 6);
12    // → ' abc '
```

_.startCase: Viết hoa các chữ cái đầu mỗi từ trong string, khá hay

_.startsWith, _.endsWith: Kiểm tra chuỗi đưa vào có phải nằm ở đầu/cuối string cần so sánh hay ko

```
1     _.startCase('--foo-bar');
2     // → 'Foo Bar'
3     _.startCase('fooBar');
4     // → 'Foo Bar'
```

```

5
6     _.startsWith('abc', 'a');
7     // → true
8     _.startsWith('abc', 'b');
9     // → false
10
11    _.endsWith('abc', 'c');
12    // → true
13    _.endsWith('abc', 'b');
14    // → false

```

_.trim, _.trimLeft, _.trimRight: Bỏ khoảng trắng

_.trunc: Cắt string và thêm ... vào cuối string nếu string quá dài (Chức năng khá hay)

_.words: Cắt các từ trong string, bỏ vào 1 mảng

```

1     _.trim(' abc ');
2     // → 'abc'
3
4     _.trunc('hi-diddly-ho there, neighborino', 24);
5     // → 'hi-diddly-ho there, n...'
6
7     _.words('fred, barney, &amp; pebbles');
8     // → ['fred', 'barney', 'pebbles']

```

3. Một số function của object

_.functions: Trả về tên toàn bộ những function của 1 object

```

1     //Trả những function của lodash
2     _.functions(_);
3     // → ['after', 'ary', 'assign', ...]

```

_.has: Kiểm tra xem 1 object có property, hoặc function với tên truyền vào hay ko

_.keys: Trả về tên toàn bộ những thuộc tính của 1 object

```

1     var object = { 'a': 1, 'b': 2, 'c': 3 };
2

```

```

3     _.has(object, 'b');
4     // → true
5
6     function Foo() {
7         this.a = 1;
8         this.b = 2;
9     }
10
11     Foo.prototype.c = 3;
12
13     _.keys(new Foo);
14     // → ['a', 'b'] (iteration order is not guaranteed)

```

_.pick: Chỉ lấy 1 thuộc tính của object

_.omit: Bỏ 1 thuộc tính của object, lấy toàn bộ những thuộc tính còn lại

```

1     var object = { 'user': 'fred', 'age': 40 };
2
3     _.pick(object, 'user');
4     // → { 'user': 'fred' }
5
6     _.omit(object, 'age');
7     // → { 'user': 'fred' }

```

4. Một số function của array

Như đã nói, lodash hỗ trợ rất nhiều hàm xử lý array, giúp việc xử lý array đơn giản hơn rất nhiều. Một lưu ý trong lodash, có 2 kiểu để gọi hàm với array như sau:

```

1     var users = [
2         { 'user': 'barney', 'age': 36, 'active': true },
3         { 'user': 'fred',   'age': 40, 'active': false }
4     ];

```

```

5      //Kiểu 1, gọi như 1 function bình thường
6      // _.tên hàm(tên mảng, các tham số khác)
7      _.filter(users, 'active', false)
8      //[{ 'user': 'barney', 'age': 36, 'active': true }];
9
10     //Kiểu 2, wrap array vào rồi gọi hàm
11     // _(tên mảng).tên hàm(các tham số khác)
12     _(users).filter('active', false).value()
13     //[{ 'user': 'barney', 'age': 36, 'active': true }];
14
15     //Mình khuyên dùng kiểu gọi 2, vì ta
16     //có thể thực hiện chaining (gọi nhiều hàm lần lượt)
17     //Tuy nhiên ta phải thêm value() ở cuối chain
18     _(users).filter('active', false).pluck('user').value();
19     //[{'user': 'barney' }]
20

```

Dưới đây là một số hàm xử lý array khá hữu dụng:

_.difference: Lấy những phần tử khác nhau giữa 2 array

_.intersection: Lấy những phần tử giống nhau giữa 2 hoặc nhiều array

_.union: Hợp 2 hoặc nhiều array lại với nhau

```

1     _.difference([1, 2, 3], [4, 2]);
2     // → [1, 3]
3
4     _.intersection([1, 2], [4, 2], [2, 1]);
5     // → [2]
6
7     _.union([1, 2], [4, 2], [2, 1]);
8     // → [1, 2, 4]

```

_.findIndex: Tìm index của phần tử đầu tiên thỏa điều kiện

_find: Tìm 1 phần tử trong array

_first, _last: lấy phần tử đầu tiên và cuối cùng của array

```
1      var users = [
2        { 'user': 'barney', 'active': false },
3        { 'user': 'fred', 'active': false },
4        { 'user': 'pebbles', 'active': true }
5      ];
6
7      _.findIndex(users, function(chr) {
8        return chr.user == 'barney';
9      });
10     // → 0
11
12     var users = [
13       { 'user': 'barney', 'age': 36, 'active': true },
14       { 'user': 'fred', 'age': 40, 'active': false },
15       { 'user': 'pebbles', 'age': 1, 'active': true }
16     ];
17
18     _.result(_.find(users, function(chr) {
19       return chr.age < 40;
20     }), 'user');
21     // → 'barney'
22
23     _.first([1, 2, 3]);
24     // → 1
25
26     _.last([1, 2, 3]);
27     // → 3
```

_.flatten: Làm dẹp array (Nếu array có chứa array thì cho ra 1 array duy nhất)

_.shuffle: Tương tự như xóc bài, tạo ra 1 array mới, chứa các phần tử của array cũ theo thứ tự ngẫu nhiên

```

1   _.flatten([1, [2, 3, [4]]]);
2   // → [1, 2, 3, [4]];
3
4   // using `isDeep`
5   _.flatten([1, [2, 3, [4]]], true);
6   // → [1, 2, 3, 4];
7
8   _.shuffle([1, 2, 3, 4]);
9   // → [4, 1, 3, 2]

```

_.some, _.any: Tương tự như every và some mình đã giới thiệu ở [bài trước](#).

Trong phạm vi bài viết, mình chỉ giới thiệu được một số hàm cơ bản cũng như cách dung cơ bản của lodash. Trong quá trình sử dụng, các bạn có thể tìm hiểu thêm, cũng như chia sẻ những gì mình tìm được ở đây nhé.

70 điều các developer giỏi thuộc nằm lòng – Phần 1

Posted on 28/05/2015 by Phạm Huy Hoàng

Ở bài trước, trong số sách đã giới thiệu, mình giới thiệu 1 cuốn sách gọi **The Pragmatic Programmer**. Như mình đã quảng cáo, cuốn sách này tập hợp rất nhiều kinh nghiệm được các tiền bối đúc kết lại qua bao nhiêu năm phát triển phần mềm. Vì sách khá dài, để tiện cho mọi người đọc, bác Jeff, chủ blog [codinghorror](#) đã rút gọn cuốn sách thành 70 điều dưới đây. 90% những điều này sẽ giúp ích cho sự nghiệp lập trình của bạn, do đó mình mạn phép dịch ra để chia sẻ lại. Bạn nào muốn có thể xem link gốc của cuối blog.

Danh sách 70 bài học đúc kết được từ **The Pragmatic Programmer**:

1. **Code có lương tâm:** Nếu đã quyết định lựa chọn sự nghiệp developer, hãy có bằng hết súc, đừng code dối hoặc làm xong cho qua việc.
2. **Code có suy nghĩ:** Hãy sử dụng đầu óc khi code. Biết tự đánh giá code của mình, khen khi code hay, chê khi code xấu.
3. **Đưa ra lựa chọn, đừng kiểm cớ:** Đừng kiểm cớ theo kiểu: “cái này không làm được”, hãy đưa ra lựa chọn kiểu : “Có thể ko làm được A, thay vào đó ta có thể làm B,C”.
4. **Đừng bỏ qua những lỗi vụn vặt:** Khi gặp các design tệ, code bựa,... hãy sửa ngay. Để lâu ngày, những lỗi nhỏ trong code sẽ tăng lên theo cấp số nhân.
5. **Sẵn sàng thay đổi:** Nếu không thể bắt mọi người thay đổi, hãy chỉ cho họ thấy những điều cần làm trong tương lai, cũng như nhờ họ góp sức thực hiện.
6. **Giữ cái nhìn toàn cục:** Đừng quá để tâm tới chi tiết, hãy giữ cái nhìn tổng thể.
7. **Chất lượng cũng là requirement:** Hãy để chính người dùng góp sức vào việc đánh giá chất lượng của phần mềm.
8. **Không ngừng học hỏi:** Hãy biến việc học (ngôn ngữ mới, kiến trúc mới ...) thành thói quen.
9. **Đừng tin những gì mình nghe và đọc:** Đừng chạy theo số đông, chạy theo công nghệ. Hãy lựa chọn, phân tích dựa trên kiến thức bản thân + cấu trúc dự án.
10. **Điều bạn nói và cách bạn nói đều quan trọng như nhau:** Không cần biết bạn điều bạn nói hay tới mức nào, nếu bạn không biết cách diễn đạt nó ra, tất cả chỉ là vô nghĩa.
11. **Đừng lặp lại:** Với code cũng như document, hãy tập trung chúng lại **1 và chỉ 1 chỗ**.
12. **Tái sử dụng:** Tìm mọi cách để thứ bạn làm ra dễ dàng tái sử dụng (tạo library, v...v), mọi người sẽ tái sử dụng nó.

13. **Component nên tách biệt:** Mỗi component nên được thiết kế chỉ nhằm 1 mục đích duy nhất, không nên phụ thuộc vào component khác.
14. **Mọi thứ luôn thay đổi:** Mọi thứ đều có thể thay đổi: Requirement, design, công nghệ sử dụng hãy luôn sẵn sàng cho những thay đổi đó.
15. **Thử và sai:** Đôi khi requirement, kiến trúc chưa rõ ràng, đừng ngại thử và sai. Nếu bạn có sai, ít ra nó cũng sẽ đưa bạn đến gần cái đúng.
16. **Dựng prototype để tìm hiểu:** Như trường hợp trên, đôi khi ta dựng 1 prototype để tìm hiểu công nghệ, hoặc tìm hiểu requirement. Thử ta thu được là kiến thức ta đã tìm hiểu, chứ không phải là prototype đó.
17. **Lập trình theo từ ngữ chuyên ngành:** Khi lập trình một hệ thống, hãy làm cho code và design của bạn giống từ ngữ thuộc lĩnh vực của hệ thống đó. VD như khi code hệ thống cho 1 trường học, ta dùng các class teacher, student, cho rạp chiếu phim thì dùng class cinema, ticket.
18. **Tập ước lượng:** Hãy thử ước lượng đánh giá thời gian, công sức bỏ ra khi code. Quá trình ước lượng này sẽ giúp bạn phát hiện 1 số vấn đề có khả năng phát sinh.
19. **Ước lượng dựa theo code:** Sau 1 thời gian code, bạn sẽ có thể ước lượng thời gian code chính xác hơn, dựa theo các số liệu trong quá khứ.
20. **Lưu trữ lại kinh nghiệm và kiến thức:** Trong quá trình code, team và bạn sẽ học được 1 số kinh nghiệm, 1 số trò “chích choá” để code chạy. Hãy lưu những kinh nghiệm này dưới dạng văn bản, trong tương lai nó sẽ có ích khi tìm lỗi, hoặc hướng dẫn thành viên mới.
21. **Hãy học cách dùng command shell:** Hầu như mọi chương trình: Window, Visual Studio, Git, v...v đều có cửa sổ command. Các cửa sổ này khá mạnh mẽ, có thể thực hiện những chức năng mà UI ko làm được thông qua các câu lệnh.
22. **Sử dụng Text Editor thành thạo:** Bạn có thể code bằng nhiều editor: Notepad++, VIM, VS,... không cần biết nó là editor nào, hãy học cách dùng nó thành thạo (Sử dụng hotkey, v...v).
23. **Luôn luôn sử dụng Source Code Control (Git, SVN):** Hiện tại 90% bạn nào cũng biết cái này rồi, ko cần nhắc lại.
24. **Gặp bug thì fix:** Không cần biết bug là lỗi của bạn hay của ai khác, nếu nó cần được fix thì hãy fix trước đã, truy cứu trách nhiệm tính sau.
25. **Dùng não khi debug:** Khi debug, nhất là những lúc bị deadline dí, ta thường hay rối. Hãy HÍT SÂU, THỞ NHẸ, suy đoán nguyên nhân gây bug, sau đó bắt đầu tìm.

26. **Nguyên nhân gây lỗi:** Khả năng rất thấp là lỗi nằm ở hệ điều hành hoặc compiler, cũng như cái library (Điều này hiện tại ko đúng lắm, vì open-source mọc lên như nấm, không test hết được). Do đó, khi fix lỗi, hãy tập trung vào code do team mình tự code trước.
 27. **Đừng dự đoán, hãy chứng minh:** Bạn đoán rằng transaction sẽ chạy mất 3 giây, database phục vụ 1000 users ... Đừng dự đoán suông, hãy dùng code, chạy test, đo đạc, tính thời gian để chứng minh điều bạn đoán.
 28. **Học một ngôn ngữ xử lý text:** Hằng ngày bạn làm việc với text, sau không viết code để máy tính xử lý chúng giúp bạn.
 29. **Viết code để tạo code:** Tương tự ý trên, nếu thấy code mình viết lặp đi lặp lại, sau ko tạo 1 chương trình tự sinh code (Hiện tại các IDE như Visual Studio, Eclipse đều hỗ trợ việc tự sinh code này).
 30. **Không có phần mềm hoàn hảo:** Không có phần mềm nào hoàn hảo. Tuy vậy, hãy giảm thiểu những lỗi mà code hoặc người dùng có thể gặp phải
 31. **Design và làm theo design:** Hãy chắc chắn rằng code của bạn **chỉ làm những điều mà nó phải làm**, ko hơn không kém. (VD code gửi mail active thì đừng block account, code mua hàng thì đừng block thẻ của người dùng).
 32. **Chương trình nên crash:** Một chương trình crash sẽ gây thiệt hại, tuy nhiên thiệt hại không nhiều bằng một chương trình chạy sai, chạy nhầm.
 33. **Sử dụng Assert:** Dùng Assert để tránh những lỗi có thể xảy ra trong code (Assert là một kiểu giống như throw Exception, ngày xưa thường hay dùng. Gần đây thì mình chỉ thấy Assert trong unit test).
 34. **Sử dụng Exception:** Chỉ dùng Exception đúng lúc, đúng chỗ, dùng lung tung sẽ khiến code thành spaghetti code.
 35. **Hãy code “có đầu có cuối”:** Nhớ giải phóng resource sau khi bạn đã dùng xong (Mở connection thì phải đóng, lấy bộ nhớ thì phải clear). Hiện tại 1 số ngôn ngữ bậc cao đã tự làm việc này, tuy nhiên lâu lâu nhớ + cẩn thận vẫn hơn.
- 35 điều tiếp sau sẽ được đăng trong [phần 2](#) nhé, mong các bạn đón xem.
Bản gốc: <http://blog.codinghorror.com/a-pragmatic-quick-reference/>

Hãy biết nói KHÔNG

Posted on 26/05/2015 by Phạm Huy Hoàng

Mình từng post một comic khá đơn giản về chuyện “[Thiết kế của một trang web có thể trở nên banh chành như thế nào](#)“. Đó là một câu chuyện về ảnh hưởng của khách hàng đã làm “banh chành” một sản phẩm của designer. Tuy chỉ là chuyện hài, nhưng nó vốn là một chuyện buồn có thật mà những người trong nghề như chúng ta đều gặp phải. Tuy nhiên, thật ra ai có lỗi trong chuyện này ...?

Sự khác biệt giữa “người lao động chân tay” và “chuyên viên” là ở chỗ: người lao động làm theo lệnh của cấp trên, còn chuyên viên thì đưa ra hướng giải quyết cho cấp trên. Người lao động được thuê để làm việc theo chỉ dẫn, còn chuyên viên được trả tiền để đưa ra chỉ dẫn **đúng đắn**.

**Khi chọn cách dịch này, mình không có ý coi thường hay xem nhẹ việc lao động chân tay nhé, mình chỉ dùng từ như vậy để dễ so sánh 2 đặc thù công việc khác nhau thôi.*



Hãy tượng tượng 1 cuộc nói chuyện giữa bệnh nhân và bác sĩ :

Bệnh nhân: Tôi đau tay.

Bác sĩ: Giờ anh muốn tôi làm gì?

Bệnh nhân: Làm tay tôi hết đau

Bác sĩ: Để tôi cắt tay anh là xong nhé? Để làm

Bệnh nhân: Không, tôi chỉ muốn hết đau thôi

Bác sĩ: Cắt hết dây thần kinh nối tới tay anh là xong, hết đau ngay

Bệnh nhân: Còn cách nào nhẹ nhàng hơn ko?

Bác sĩ: Xin lỗi, hết giờ làm rồi

Dĩ nhiên bác sĩ bình thường không hành xử thế này. Dù bệnh nhân là khách hàng, bệnh nhân trông cậy vào bác sĩ đưa ra câu trả lời, phương pháp giải quyết

Một phiên bản khác của cuộc nói chuyện này

Bệnh nhân: Tôi muốn chặt tay.

Bác sĩ: Tay anh bị gì?

Bệnh nhân: Đau vл, mệt lấm rồi, cắt cmn đї

Bác sĩ: Dưa tay tôi xem? Chắc bị bong gân hay trật khớp thôi. Phải chụp X-quang.

Bệnh nhân: Không, cắt cmn đї

Bác sĩ: Xin lỗi anh, tôi không chặt tay lành lặn

Bệnh nhân: Nhưng tôi trả tiền mà. Tôi nói gì ông phải làm chit?

Bác sĩ: Không. Nếu chặt tay anh, tôi sẽ vi phạm đạo đức của một người hành nghề y.

Bạn muốn làm một bác sĩ thế nào? Quay lại ngành của chúng ta, bạn muốn là một developer như thế nào?

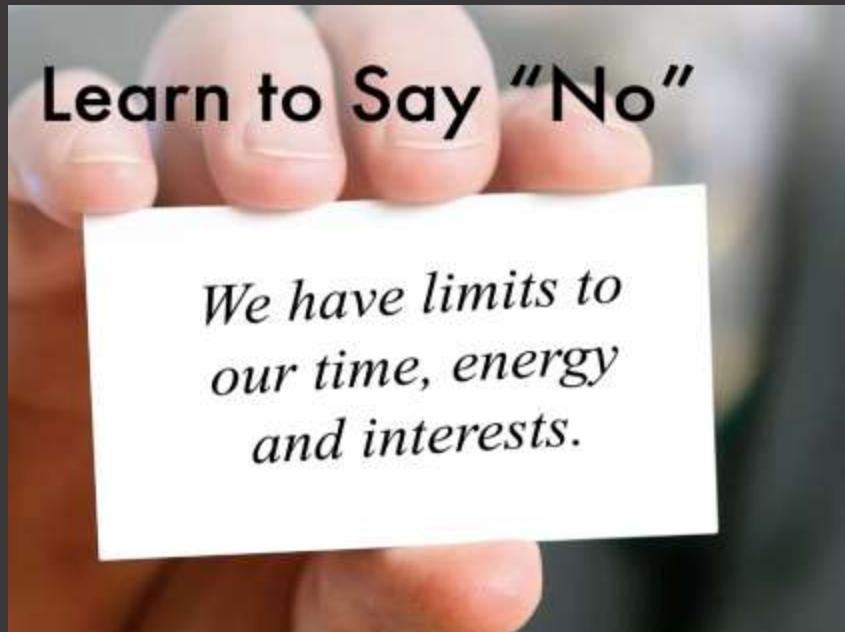
Lập trình viên, cũng là “chuyên viên”. Bạn có nhiều kiến thức về thiết kế, hiện thực phần mềm hơn khách hàng/cấp trên của bạn. Bạn được trả tiền cho chuyện đó. Tất cả quy kết lại một điều. Chuyên viên là người **phải dám nói KHÔNG**. Khi cấp trên/khách hàng đưa ra 1 định hướng, yêu cầu vô lý, nhiệm vụ của bạn là phải từ chối yêu cầu đó. Có nguy hiểm gì không? Đương nhiên là có. Nhưng, là một chuyên viên, đồng nghĩa với việc bạn phải dám giữ lập trường của mình. Có những chuẩn mực mà người chuyên viên **không bao giờ nên vi phạm**.

Đĩ nhiên, nói KHÔNG chỉ là một phần. Là một chuyên viên, bạn phải sử dụng kỹ năng của mình, đưa ra những lựa chọn thay thế khả thi, phù hợp. Người chuyên viên phải biết *đàm phán* với cấp trên, để đưa ra những định hướng, giải pháp thỏa mãn yêu cầu của cả hai bên.



Trong mẩu truyện mình đã đăng, chàng web-designer đã **không hành xử như một chuyên viên**. Anh làm việc như một “người lao động tay chân”. Đống shit ở cuối truyện là do lỗi của anh. Lê ra anh phải biết **nói KHÔNG**, và đàm phán/giải thích với khách hàng, thay vì làm mọi việc khách hàng yêu cầu. Trong truyện, chàng designer là nạn nhân – giỏi giang nhưng yếu thế, còn khách hàng là một gã ngu đần lại còn lạm quyền. Sự thật thì, anh designer đã tự biến mình thành nạn nhân, chối bỏ trách nhiệm, không chịu nói KHÔNG trước những yêu cầu vô lý của khách hàng.

Là một lập trình viên/chuyên viên, bạn *đừng bao giờ* để mình trở thành một nạn nhân như vậy nhé.



Bài viết được lược dịch từ
trang :<https://sites.google.com/site/unclebobconsultingllc/blogs-by-robert-martin/saying-no>. Tác giả bài viết là Robert Martin, người viết cuốn sách [Clean Code](#). Ông cũng là một cây đa cây đề trong ngành phần mềm.

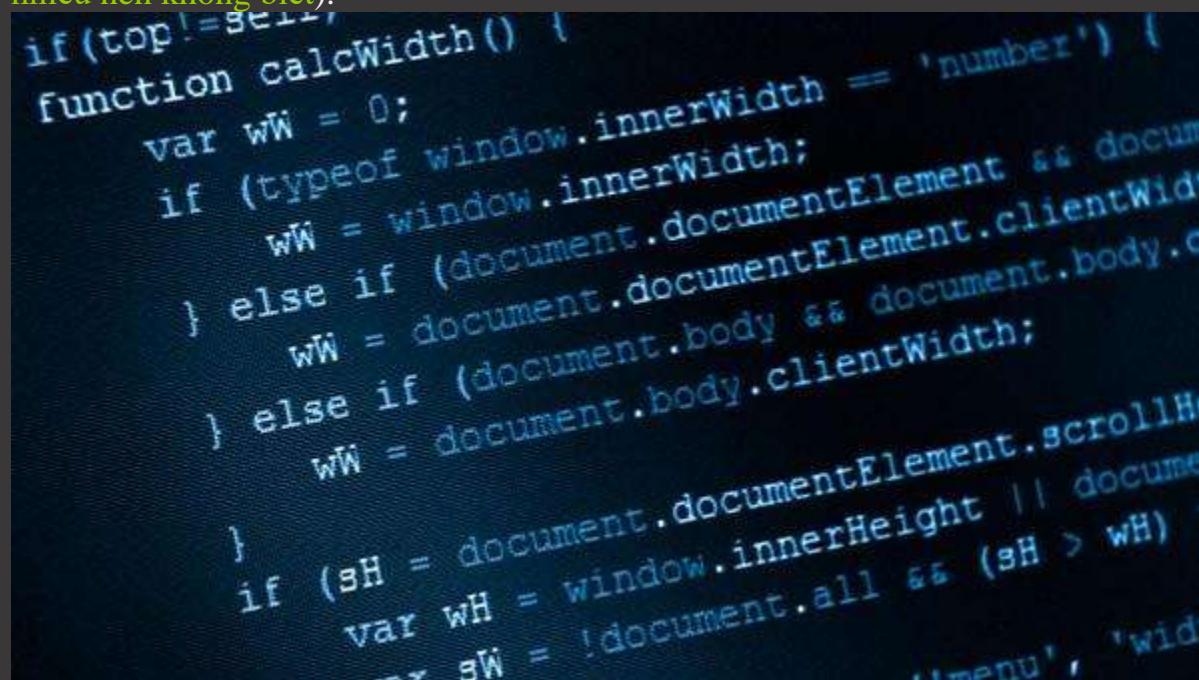
Éo ai quan tâm đến code bạn viết đâu

Posted on 21/05/2015 by Phạm Huy Hoàng

Lưu ý: Bài viết này được phỏng dịch từ 1 blog tiếng Anh. Có khá nhiều bạn sẽ không đồng ý với ý kiến được đưa ra trong bài viết này, bản thân mình cũng cảm thấy có đôi chỗ chưa đồng tình lắm (Sẽ comment bằng chữ màu xanh). Tuy nhiên, những ý kiến cũng như dẫn chứng đưa ra trong bài viết – theo ý mình – là khá chính xác. Mong rằng mọi người có thể rút ra được vài điều bổ ích (như mình) sau khi đọc bài này. Bạn có thể tham khảo bài viết gốc tại

đây: <http://mortoray.com/2015/04/20/nobody-cares-about-your-code/>

Có một sự thật đón đau ít người biết về nghề lập trình. Lập trình viên thường bỏ công sức để trau chuốt code của mình trở nên hoàn hảo mà quên 1 sự thật: Thứ quan trọng nhất sự không phải là code, mà là sản phẩm. Biết được điều này, lập trình viên sẽ gia tăng hiệu suất làm việc, sản phẩm do anh ta làm ra cũng sẽ được coi trọng hơn (Điều này chỉ đúng với các start-up, các project làm ra sản phẩm. Với cái project outsource, code chính là sản phẩm, bọn Nhật nhiều khi review code rất kĩ, bắt lỗi tới từng câu lệnh, cách đặt tên biến. Tác giả là người nước ngoài chắc cũng không làm outsource nhiều nên không biết).



```
if (top != self) {
    function calcWidth() {
        var wW = 0;
        if (typeof window.innerWidth == 'number') {
            wW = window.innerWidth;
        } else if (document.documentElement.clientWidth > 0) {
            wW = document.documentElement.clientWidth;
        } else if (document.body.clientWidth > 0) {
            wW = document.body.clientWidth;
        }
        if (sH = document.documentElement.scrollHeight) {
            var wH = window.innerHeight || document.documentElement.clientHeight;
            if (sW = document.all || (sH > wH)) {
                return sW;
            } else {
                return wW;
            }
        } else {
            return wW;
        }
    }
}
```

Code chỉ là công cụ

Công việc của developer chúng ta không phải là viết code, mà là sử dụng code để tạo ra phần mềm, với những chức năng mà người dùng mong muốn. Code là thứ giúp chúng ta thực hiện điều đó, nhưng xét cho cùng, nó chỉ là công cụ. Giống như công việc của thợ cắt tóc không phải là sử dụng kéo (kéo = code), công việc của họ là tạo ra thứ gì đó (một mái đầu đẹp) với cây kéo.

Tuy vậy, một số quy trình trong ngành phần mềm có thể làm ta lầm tưởng rằng code chính là sản phẩm. Vd, việc refactor code, có thể xem đó là 1 qui trình làm cho code tốt hơn, hoặc làm cho sản phẩm tốt hơn. Nhìn từ góc độ của khách hàng hoặc người quản lý, refactor code là một hành động vô bổ, còn có thể làm nảy sinh bug mới, tốn thời gian và công sức.

Sản phẩm chúng ta tạo ra (phần mềm) được đánh giá qua những gì người dùng nhìn thấy, chúng ta cần tự nhắc nhở mình điều này mỗi khi viết một dòng code. Nếu phần mềm nhảm chán, không đáp ứng đủ chức năng, người dùng sẽ **éo quan tâm** đến những thứ đằng sau, toàn bộ những thứ đẹp đẽ tuyệt vời chúng ta làm trong code đều trở nên vô nghĩa.

Điều đó không có nghĩa là code hoàn toàn vô giá trị, có thể thoái mái thả nổi chất lượng. Những vấn đề như bảo mật, validation, khả năng chịu lỗi cần phải được giải quyết bằng cách code đúng cách, sử dụng đúng thư viện, đó cũng chính là công việc của lập trình viên chúng ta. Code chất lượng, dễ hiểu thì người khác sẽ review dễ dàng hơn (**Theo** mình, ý của tác giả ở đoạn này là: tuy khách hàng không quan tâm đến code của bạn, nhưng còn 1 số người khác như developer, technical lead sẽ khá quan tâm đến nó).

Tập trung vào chức năng

Hãy tưởng tượng bạn đang ở trong một cuộc họp trực tiếp với khách hàng, hoặc product owner. Chúng ta báo cáo gì với họ? Chúng ta có nói rằng đã tạo được bao nhiêu class, viết được bao nhiêu dòng code ko? Hay chúng ta nói cho họ nghe về cách chúng ta sửa lỗi ứng dụng, script để deploy, hoặc ta thiết kế database như thế nào.

Tiếc thay, câu trả lời thường là **CÓ**. Hãy nhìn mặt khách hàng lúc này, trông họ chan chán, ngu ngơ như con cá bơi, hoặc giả bộ gật gù chờ hết chuyện. Chúng ta đi vào quá nhiều chi tiết **vô giá trị đối với khách hàng**. Không phải học không biết, không hiểu, mà đơn giản là họ **éo quan tâm** (Ví dụ dễ thấy, nếu bạn nhận làm web quảng cáo cho 1 công ty, họ chỉ cần sản phẩm là 1 trang web kèm tên miền, họ không quan tâm bạn

dùng PHP hay Java hay C#, bạn viết HTML4 hay HTML5, họ không biết, và có biết cũng không thèm quan tâm).

Hãy tượng tượng bạn có 1 đội designer trong công ty. Trong buổi họp, bạn phải nghe bọn designer làm nhảm về photoshop, về việc tại nó đã tạo layer thế nào, thiết kế gradient tương phản ra sao, tại nó viết script xử lý ảnh như thế nào. Chúng ta **éo quan tâm**, chúng ta chỉ cần biết : Khi nào bọn mài design xong để tại tao còn code.

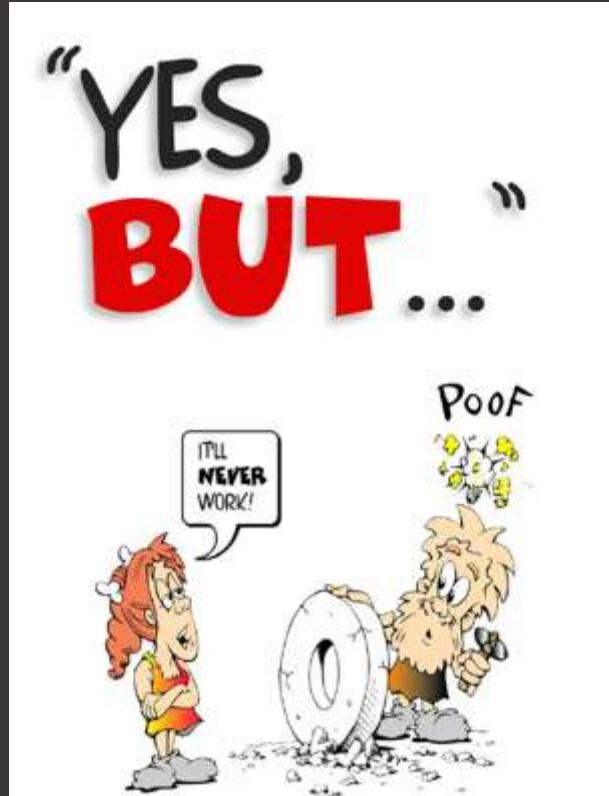
Chúng ta nên tập trung vào chức năng khi báo cáo với khách hàng: chức năng tạo báo cáo đã xong, nhưng còn thiếu phần ABC; trang web đã hiển thị rõ đẹp trên di động, nhưng cần chỉnh sửa chút đỉnh trên tablet. Đây là những thông tin mà quản lý và khách hàng quan tâm, nó giúp họ hiểu rõ tiến độ của dự án, cho họ thêm thông tin trong việc lên kế hoạch



Mấy cái library vô dụng

Một câu hỏi đặt ra dành cho các bạn lập trình viên: Đã bao giờ các bạn chọn 1 library để sử dụng dựa trên source code của nó chưa? Mình thì chưa. Vd như mình cần tìm 1 library hỗ trợ việc resize ảnh trong C#, mình sẽ google, sau đó xem library nào được nhiều người sử dụng nhất, chức năng có nhiều hay không, cách sử dụng có dễ hay không. Thứ mình quan tâm là gì, chính là **sản phẩm, không phải code của thư viện đó**. Nếu 1 thư viện được code rất có bài bản, áp dụng design pattern, interface, DI, nhưng mà chức năng chỉ có 1 chút, liệu bạn có chọn không? Câu trả lời đương nhiên là không.

Bạn thấy đây, **đến cả lập trình viên chúng ta còn không thèm quan tâm đến code của lập trình viên khác**, làm sao bạn có thể đòi hỏi “lũ khách hàng” quan tâm đến code của mình.



Đúng, nhưng mà...

Vì không ai quan tâm đến code, chúng ta có quyền viết code ẩu cho xong chuyện.
Không, hoàn toàn không phải. Code chất lượng thì sẽ cho ra sản phẩm chất lượng.
Dù không ai quan tâm đến code, để giải quyết vấn đề ta vẫn phải dùng đến code. Quay lại chuyện cái kéo ở ban đầu, kéo bén hay kéo cùn thì đều cắt ra tóc được, nhưng xài kéo rỉ sét (code quá tệ) thì sẽ hại cả thợ lẫn người cắt. Công việc của lập trình viên chúng ta là **chăm chút cho sản phẩm, không phải cho code**. Hãy thử đặt mình vào vị trí của quản lý, hoặc khách hàng, tập trung vào chức năng, chứ không phải công cụ. Chức năng, chứ không phải code, mới là thứ mang lại giá trị thực sự cho sản phẩm ta làm ra.

Thiết kế của một trang web trở nên “banh chành” như thế nào

Posted on 19/05/2015 by Phạm Huy Hoàng

Đi lòng vòng tìm được một cái comic khá hay về web designer ở đây: http://theoatmeal.com/comics/design_hell.

Chợt nhớ ra mình cũng biết chút đỉnh về Photoshop nên dịch ra để chia sẻ cho các bạn luôn. Nếu các bạn thấy hay mình sẽ cập nhật thể loại này nhiều hơn.



Khách hàng cho bạn xem trang web hiện tại

Cả hai vừa cười vừa chê trang web cũ



Chỉ có vài thay đổi "nho nhỏ"

Desogn hết xẩy. Nhưng vì anh là CEO nên anh phải yêu cầu chú sửa vài chỗ để thể hiện trách nhiệm, Còn nữa, anh thích dùng mấy câu như "trải nghiệm người dùng", "tiếp cận giao tiếp" cho nó có vẻ nguy hiểm mặc dù anh chẳng biết cái vẹo gì về vi tính

Chú mày làm cái design "cứng" hơn tí được không. Góc cạnh thêm 1 chút. Anh thấy có vẻ ko ổn. *

*Lời tác giả: Khách hàng thật sự từng nói vậy với tôi. Tới ngày hôm nay tôi vẫn éo biết thế nào là design "cứng" hay "góc cạnh". Tôi cũng éo biết design dựa theo "cảm giác không ổn" của khách hàng.

Thay đổi nhỏ nhở dần dần nhiều lên

Dần dần chúng không-còn-nhỏ-nữa



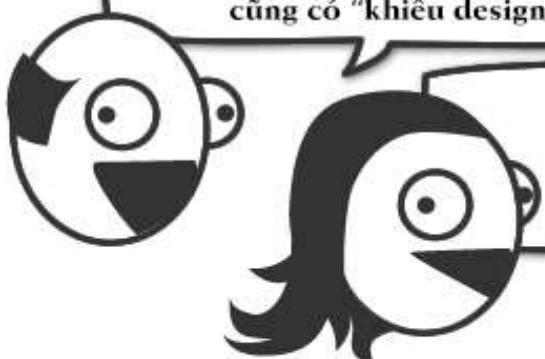
Anh nghĩ lại rồi, chắc chắn phải chuyển font sang Comic Sans. Chú nhớ làm thêm vài cái hiệu ứng long lanh, mới giống web 2.0

Còn nữa, cho các trang nó bớt "thẳng thớm" lại đi, tao nhìn vào thấy toàn dòng kè. *

*Lời tác giả: Khách hàng thật sự từng nói vậy với tôi. Trong design không có đường thẳng nào cả, ý khách hàng là các hình chữ nhật tạo ra bởi thẻ `<div>` và `<p>`

Khách hàng kéo người vào góp ý

"Trông tuyệt vời đấy, nhưng anh muốn lấy ý kiến từ bạn bè, đồng nghiệp, chú chó, chú mèo v...v"



Anh nhờ mẹ anh góp ý luôn này.
Ngày xưa bà từng design mấy cái tờ rơi quảng cáo từ năm 1982, nên cũng có "khiếu design" lắm. *

Cái design chú làm cần thêm vài màu cho nó sáng sủa, nhfn u ám quá. Thêm tí màu hường nhá. Thêm mấy con mèo vào luôn. AI CŨNG THÍCH MÈO HẾT

*Lời tác giả: thật sự tôi từng gặp một khách hàng mời cả mẹ mình tham gia vào quá trình design để góp ý và phê bình

Không còn hi vọng gì nữa

Bạn bắt đầu muốn chuyển qua làm một số nghề thanh thản hơn
như hút hầm cầu, khoan cắt bê tông, bán keo dính chuột...



Con chó của anh, con Kiki, nó là một người bạn thân thiết trong đời anh. Anh muốn chia sẻ thêm vào vài "điểm nhấn", đưa lên web, như là con Kiki đang nói chuyện với người dùng ấy. Anh sẽ gửi chú những điều mà con Kiki nó hay suy nghĩ, dạng như "Mình thích xương" hay "Chào mừng đến với trang web, tôi là cún đây, có muốn bắt tay không". LOL *

*Lời tác giả: Anh không hề bịa - Thật sự từng có khách hàng ra yêu cầu như thế. Lần đầu muốn bóp cổ cắt tr*t m một khách hàng đến vậy .

Bạn không còn là một designer nữa

Trong mắt khách hàng, bạn chỉ là con chuột trong một chương trình đồ họa, có thể điều khiển bằng cách ra lệnh, gửi mail hay chat



phì phò phì phò, tao cũng làm web
được này! phò phò phì phì *

*Lời tác giả: Tôi từng gặp một khách hàng sử dụng chính design của tôi, chỉnh lại bằng photoshop, rồi gửi lại cho tôi phiên bản đã sửa. Sau bản chỉnh sửa thứ 13 tôi chính thức cắt cmn hợp đồ ngluôn.

Một đống shit ra đời

Khách hàng hoàn toàn quên rằng họ thuê bạn, một designer, để thiết kế một sản phẩm tuyệt vời cho họ.

Nếu bạn là kĩ sư thiết kế động cơ máy bay, chẳng biết khách hàng có dám can thiệp nhiều vậy không nhỉ?



11 điều luật mà mọi lập trình viên nên tuân theo

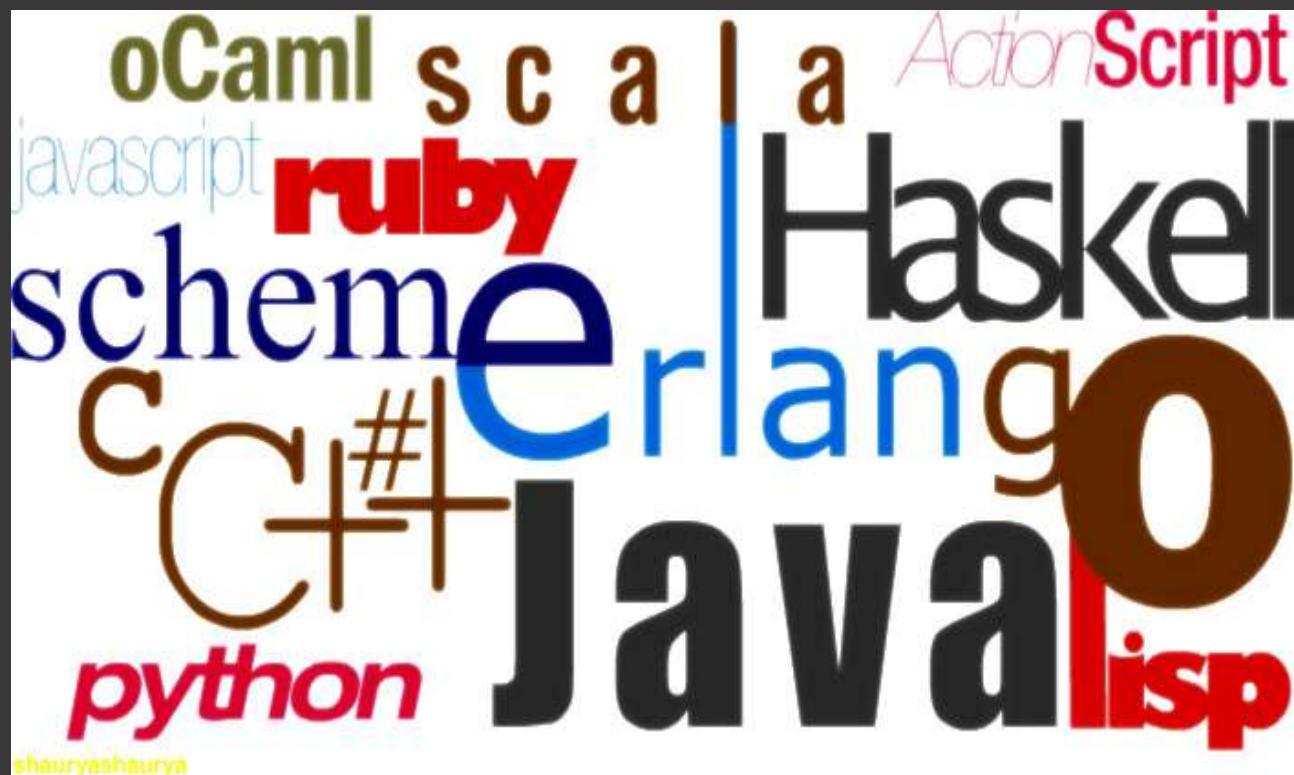
Posted on 12/05/2015 by Phạm Huy Hoàng

11 điều luật này được phỏng dịch từ bài viết của John Sonmez – chủ nhân của blog <http://simpleprogrammer.com/>, một blog khá nổi tiếng vì IT. Anh nổi tiếng với các bài viết đơn giản, dễ hiểu nhưng vô cùng hữu ích, không chỉ về code mà còn về con đường phát triển của lập trình viên. Những bài viết của anh trên blog đã tạo cảm hứng cho mình tạo nên blog này.

1. Công nghệ chỉ là một PHƯƠNG PHÁP để giải quyết vấn đề, không phải là GIẢI PHÁP

Cộng đồng lập trình viên là một cộng đồng khá phù phiếm, chúng ta chạy theo đú thử công nghệ đang hot: NodeJS, AngularJS, NoSQL ... Mình không nói chuyện này là xấu, tuy nhiên đôi khi chúng ta quá chú trọng vào công nghệ mà quên mất rằng nó chỉ là 1 phương pháp giải quyết vấn đề.

Đơn cử, nếu làm một trang web mua bán đơn giản, ít người sử dụng, ta có thể dễ dàng sử dụng PHP & MySQL, hoặc sử dụng các framework có sẵn,... không cần phải cố gắng áp dụng công nghệ hầm hố để giải quyết 1 vấn đề đơn giản.



2. Đừng code “thông minh”, hãy code rõ ràng

Có bao giờ bạn gặp trường hợp thế này: Có 1 chức năng cần sửa, hoặc 1 con bug khủng. Bạn nghĩ ra cách hiện thực/ cách giải vô cùng thông minh, ngắn gọn, tự vỗ đùi 1 phát khen mình thông minh. Tuy nhiên, 1 tháng sau, vào xem lại code (của chính mình) và không biết cái đống code “thông minh” của mình chạy như thế nào, làm sao sửa.

Chúng ta là lập trình viên, chất lượng của code được đo bằng tính rõ ràng, tính dễ bảo trì v...v, chứ không phải code càng ít dòng là càng giỏi như thời còn đi thi, đi học. Để biết cách viết code rõ ràng, các bạn có thể tham khảo cuốn [Clean Code](#) mà mình đã viết review trước đây.

3. Đừng viết code thừa, hãy viết đúng những gì cần viết

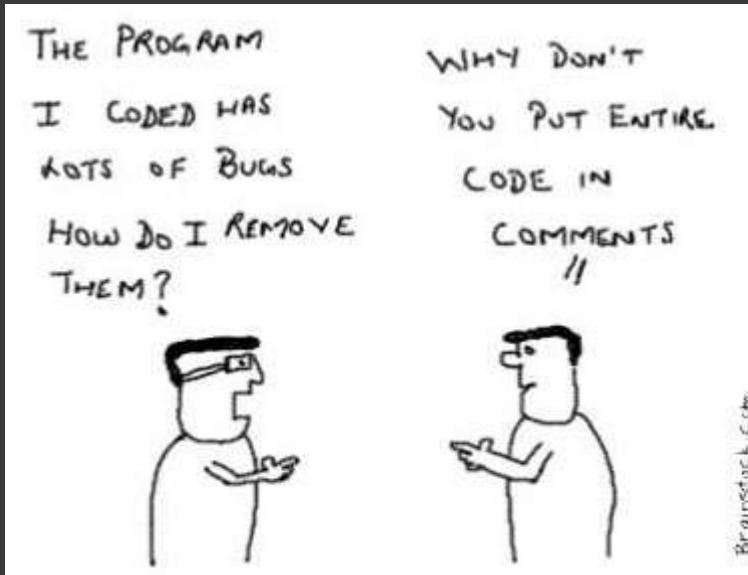
Hơi trái ngược với điều 2 nha? Điều này có nghĩa là: Bạn chỉ cần viết code để hiện thực chức năng cần làm. Có thể code sẽ hơi thừa một chút, để chỗ trống cho sau này có thể bảo trì, nâng cấp v...v. Tuy nhiên, đừng viết code thừa, code sẵn để “trong tương lai dùng tới”. Có thể trong tương lai bạn sẽ chẳng cần tới nó đâu. Ngược lại, nhiều code còn sinh ra nhiều bug hơn, làm việc bảo trì khó khăn hơn.

4. Đừng lạm dụng comment

Comment làm lập trình viên lười hơn. Thay vì đặt tên biến dễ hiểu, viết flow chương trình rõ ràng, họ viết 1 đống sh*t, sau đó dùng comment để mô tả đống sh*t đó làm gì.

Ngoài ra, khi update code, họ rất lười update comment, do đó đôi khi comment chỉ là những dòng chữ thừa, vô dụng, không có tác dụng thực tế gì.

Cá nhân mình thì cũng hạn chế việc sử dụng comment, chỉ dùng nó để mô tả những điều không mô tả được bằng code. Hãy tập code thế nào để không cần nhìn comment bạn cũng hiểu được code chạy thế nào. Tới lúc làm được điều đó, bạn đã lên một “[đẳng cấp](#)” khác.



5. Trước khi viết code, hãy xác định code sẽ làm gì

Tình huống này khá quen, mình chắc không ít các bạn đọc blog này đều gặp phải. Đôi khi chúng ta cắm đầu vào code trước khi làm rõ vấn đề. Quả thật là có nhiều lúc ta phải code mới biết cần giải quyết chuyện gì, code thế nào. Tuy nhiên, lời khuyên ở đây là: Hãy xác định mục đích code sẽ thực hiện trước khi code.

6. Hãy tự test code của mình trước khi quăng nó cho tester

Dĩ nhiên, test là trách nhiệm của tester. Tuy nhiên, là một lập trình viên “có đạo đức”, hãy tự test những case cơ bản, fix những lỗi ngu ngơ ngáo ngắn như “validation” v....v trước. Đảm bảo chất lượng là trách nhiệm của mọi người.

7. Mỗi ngày hãy học 1 điều mới

Hồng Thất Công từng nói với Hoàng Lão Tà : “Trong giới võ học, không tiến túc là lùi”. Trong giới develop học cũng thế, nếu mỗi ngày không học 1 điều mới, bạn sẽ thụt lùi. Bạn sẽ cảm thấy càng ngày mình càng già, tụt hậu về công nghệ.

Để giải quyết chuyện này, mỗi ngày hãy dành ra 15p đọc sách (Ở các bài viết sau mình sẽ giới thiệu 1 số sách hay), mỗi năm hãy tự học một công nghệ mới, một ngôn ngữ mới, bạn sẽ nhận ra học cũng là 1 niềm vui đấy.

8. Viết code là chuyện rất thú vị

Thật lòng mà nói, nghề lập trình viên là một nghề có mức lương khá ổn, công việc cũng dễ kiếm (Hãy nhìn các bạn công nhân lương 5,6 triệu/tháng, nhiều bạn kinh tế ngân hàng ra trường không có việc).

Đôi khi nhìn lại, đi làm 1 ngày 8 tiếng ngồi với code và IDE, ta bỗng cảm thấy mệt mỏi. Hãy nhớ lại những ngày đầu tập code, nhớ lại niềm vui khi những dòng code đầu tiên chạy được. Sau đó nhìn lại mình, được code (làm điều mình thích), lại còn được trả tiền, cuộc sống còn gì hơn nữa.

9. Chấp nhận rằng không phải thứ gì mình cũng biết

Đừng cố gắng học hết mọi thứ, đừng cố gắng tỏ ra mình biết mọi thứ. Bạn có thể nói “em không biết”, cũng có thể hỏi người khác khi có vấn đề không rõ.

Càng học nhiều bạn sẽ thấy có nhiều thứ mình không biết, thay vì cố gắng học hết tất cả, hãy học cách tự học, sau đó xác định những kiến thức nào **thật sự cần thiết**, và tập trung vào nó.

10. Cách giải quyết tốt nhất còn tùy vào hoàn cảnh

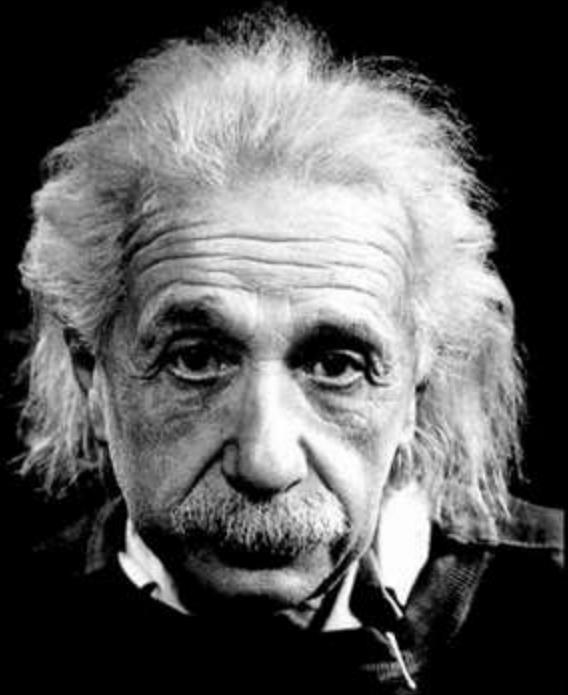
Không phải lúc nào cũng nên áp dụng 3 lớp, áp dụng IoC, áp dụng Test Driven Development. Mặc dù chúng được gọi là “best practice” – những cách giải quyết tốt nhất, chúng ta cũng phải áp dụng tùy theo hoàn cảnh, không phải cứ mù quáng áp dụng vào là được.

11. Giữ cho mọi thứ đơn giản

Cách giải hay nhất cho một vấn đề thường là cách đơn giản nhất (Đừng làm với cách giải “thông minh” mình nhắc ở trên). Tuy nhiên, cách “đơn giản” này đôi khi rất tốn công sức mới tìm ra được. Trước khi đưa ra một cách giải quyết phức tạp cho 1 vấn đề, hãy tự hỏi mình “có cách nào giải quyết nó một cách đơn giản hơn không”

“Everything should be made
as simple as possible,
but not simpler.”

Albert Einstein



11 điều luật tới đây là hết, bạn nào muốn góp ý hãy bổ sung gì cho mình hãy bổ sung trong phần comment nhé.

Bài viết gốc: <http://simpleprogrammer.com/2015/03/16/11-rules-all-programmers-should-live-by/>

Cột mốc 1000 view – Lời cảm ơn chân thành đến bạn đọc và mọi người

Posted on 07/05/2015 by Phạm Huy Hoàng

Mình bắt đầu viết blog này vào ngày 31/12 năm trước, bắt đầu bằng bài viết về bản thân và giới thiệu blog. Đây là bài viết thứ 28 của blog này, đánh dấu lượt view thứ 1000.

<https://toidicodedao.wordpress.com/2014/12/31/gioi-thieu/>



Từ đó tới nay tính ra đã được 5 tháng, lúc blog đạt được mốc 1000 view vẫn còn là tháng 4, tính ra mình đạt mốc 1000 view sau 4 tháng, không phải quá xuất sắc nhưng cũng không đến nỗi tồi. Kỉ lục của blog là hơn 100 views/ngày, khi mình post link bài viết về [Lambda Expression](#) trên group ASP.NET MVC facebook.

Nói về cái nghiệp viết blog, tính ra cũng có lầm thứ để kể. Từ lúc bắt đầu viết, mình luôn ráng giữ tiến độ 1 tuần 2 bài viết (vào **thứ 3** và **thứ 5**), để nhanh chóng tăng content cho blog. Ban đầu còn khá sung, mình luôn viết bài đầu đủ vào thứ 3, thứ 5, thứ 7 còn rảnh viết thêm bài, để có bài dự phòng đăng khi mình không kịp viết. Mỗi lần có ý tưởng, mình thường tạo bản nháp, viết sẵn tựa đề, sau đó có thời gian rảnh mới viết bài. Nhiều khi có ý tưởng, ngồi xuống viết nguyên bài luôn. Về sau dần dần hết ý tưởng, phải đi đọc blog nước ngoài, xem có thứ gì hay ho để viết. Đôi khi phải tách 1 bài ra làm 2, 3 phần để đảm bảo tiến độ 1 tuần 2 bài, bạn đọc có ức chế thì xin thông cảm.

Như đã chia sẻ từ bài đầu, mình viết blog để luyện khả năng viết, chia sẻ kiến thức cũng như tạo dựng danh tiếng. Danh tiếng thì chưa thấy đâu, lâu lâu bỏ cả tiếng để viết 1 bài mà chẳng thấy ma nào xem, kể cũng buồn + tủi thân. Lâu lâu bí đè tài mà viết, buồn chán muốn drop, mình lại nhớ tới câu “Khi bạn sắp bỏ cuộc, hay nhớ tới lý do

ban bắt đầu”. Như hôm đang viết bài này, mình vừa đi chơi với gấu về, 11h đêm, buồn ngủ đến mức mắt díp lại vẫn ráng mà gõ cho được 1 bài hoàn chỉnh. Tính ra mình cũng học được thêm nhiều thứ khi ráng viết và duy trì cái blog này.

Những dòng trên chỉ là đôi điều mình muốn chia sẻ với các bạn developer, hi vọng tương lai cộng đồng developer Việt Nam sẽ có thêm thật nhiều blog, thật nhiều kênh chia sẻ kiến thức và kinh nghiệm. Dưới đây là 1 số series bài viết được nhiều người xem trên blog của mình, mình nghĩ nó khá thú vị + sẽ giúp ích được nhiều cho các bạn.

1. Series “[Học ngôn ngữ lập trình gì bây giờ](#)”: Cái nhìn tổng quát + lời khuyên của mình về ngôn ngữ lập trình mà các bạn sinh viên nên lựa chọn để gia tăng cơ hội nghề nghiệp.
2. Series “[Lật mặt na sự bá đạo của LINQ](#)”: Series này đề cập tới những yếu tố cấu thành LINQ: [Lambda Expression](#), [Generic](#), [Extension method](#), [Delegate](#), [IEnumarable](#). Đọc xong series này, kiến thức cơ bản của bạn sẽ vững hơn nhiều, đồng thời bạn sẽ có 1 cái nhìn tổng quan và rõ ràng về sự “màu nhiệm” của LINQ.
3. Series “[C# hay ho](#)”: Bao gồm những bài viết lặt vặt, tổng hợp những điều thú vị trong C#, có thể có nhiều người biết, hoặc biết không đầy đủ. Tuy nhỏ nhặt nhưng những kiến thức này khá hữu dụng.

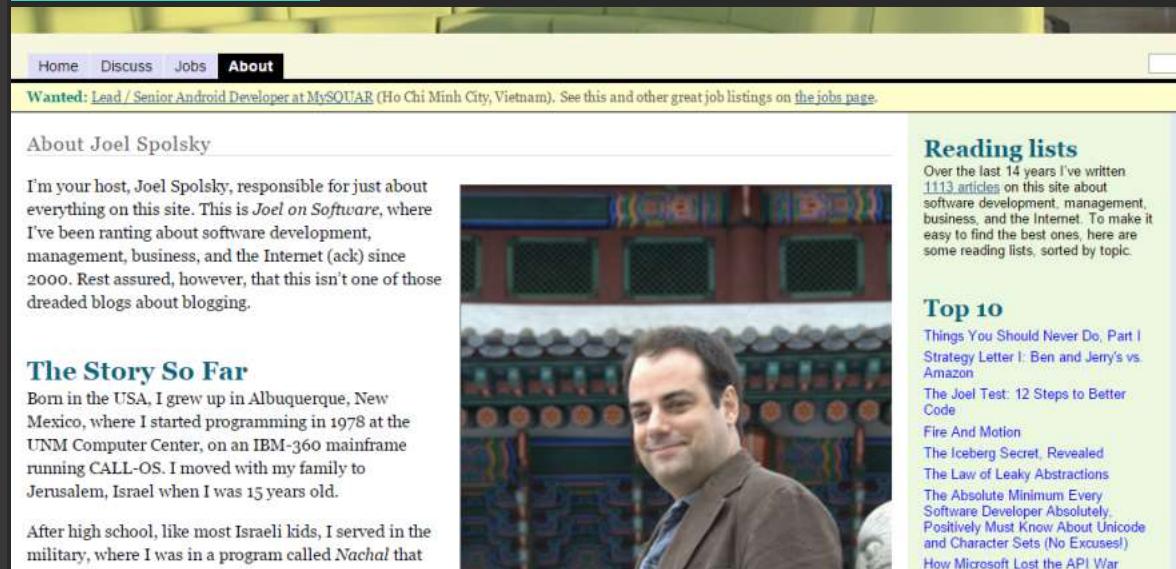
Bài viết đến đây là hết. Cột mốc tiếp theo mình đặt ra là 5000 view, hi vọng blog vẫn còn được duy trì đều đặn cho tới lúc đó. Cảm ơn các bạn đã quan tâm theo dõi.

Top 5 blog về IT đáng đọc

Posted on 05/05/2015 by Phạm Huy Hoàng

Vào những lúc rảnh rỗi, mình thường hay đọc đủ thứ: Sách, manga, tiểu thuyết, báo lá cải... Từ hồi đi làm, không còn được vào webtretho hay vozforum thoải mái, mình bắt đầu chuyển qua đọc ebook IT và blog IT xem như để giết thời gian. Đọc mấy thứ này thì người xung quanh đi qua vẫn thường bạn đang chăm chú code hay nghiên cứu, lại còn tăng khả năng + kiến thức lập trình, do đó hôm nay mình chia sẻ cho mọi người, mong được ủng hộ. Lưu ý, top 5 này chỉ là ý kiến của cá nhân mình, bạn nào muốn đóng góp thêm vào danh sách này có thể thoải mái đóng góp trong mục comment nhé.

1. Joel on Software



The screenshot shows the homepage of [Joel on Software](http://joelonsoftware.com). At the top, there's a navigation bar with links for Home, Discuss, Jobs, and About (which is highlighted). Below the navigation, a banner says "Wanted: Lead / Senior Android Developer at MySOURAR (Ho Chi Minh City, Vietnam). See this and other great job listings on [the jobs page](#)". The main content area starts with a bio for Joel Spolsky: "About Joel Spolsky". It includes a photo of him smiling. To the right, there's a "Reading lists" section with a link to 1113 articles, and a "Top 10" sidebar with links to various articles like "Things You Should Never Do, Part I" and "The Joel Test: 12 Steps to Better Code".

Chủ nhân blog này là Joel Spolsky – người sáng lập ra mạng lưới [Stackoverflow](http://stackoverflow.com). Ông viết blog của mình trong 10 năm ròng rã, kể về những trải nghiệm, những bài học rút ra được trong sự nghiệp lập trình. Mình xin kể sơ lược tiểu sử của ông: Sau khi tốt nghiệp, ông vào làm Program Manager cho Microsoft, tham gia vào dự án Excel. Sau đó ông nhảy một số công ty, và xây dựng một công ty cho riêng mình. Joel vừa là một developer giỏi, vừa là một doanh nhân tầm cỡ, cũng như một cây bút siêu hạng. Ông viết nhiều bài viết về lập trình, thiết kế, cũng như một số tâm đắc trong quá trình thành lập công ty, tuyển dụng. Dù vị trí của bạn là gì:

developer, team leader, manager, giám đốc, ... bạn sẽ tìm được nhiều điều hay trong blog của ông.

Điều đáng tiếc là Joel đã ngừng viết blog khoảng 2,3 năm gần đây, lâu lâu ông chỉ lên blog để giới thiệu công ty và phần mềm sắp ra.

2. Coding Horror

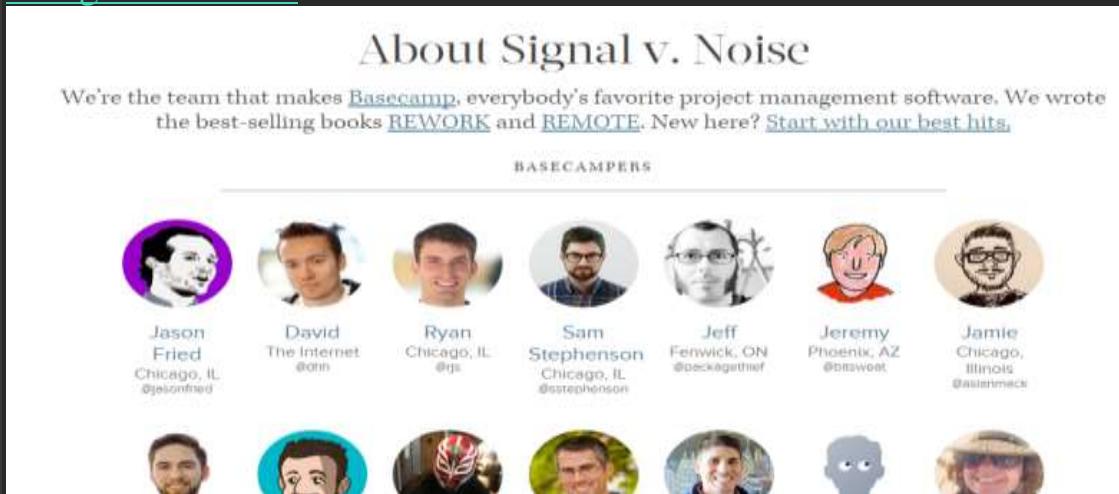


The screenshot shows the homepage of Coding Horror. At the top left is a cartoon illustration of a person with a large head and a small body. To its right is the title "CODING HORROR" in bold capital letters, with the subtitle "programming and human factors" below it. On the far right is a search bar with the text "Google Custom Search" and a magnifying glass icon. Below the header, there's a sidebar titled "RESOURCES" containing links to "About Me", "@codinghorror", "discourse.org", "stackexchange.com", and "Recommended Reading". A "Subscribe in a reader" link is also present. The main content area has a timestamp "01 Feb 2004" and a section titled "About Me". It features a yellow square icon of a house with two people inside. The text describes Jeff Atwood's life: "I'm Jeff Atwood. I live in Berkeley, CA with my wife, two cats, one three children, and a whole lot of computers. I was weaned as a software developer on various implementations of Microsoft BASIC".

Tác giả trang web này là Jeff Atwood, nhà đồng sáng lập [Stackoverflow](#). Blog này nghiêng về technical nhiều hơn so với blog của Joel. Những bài viết của Jeff không chỉ là về code, về framework mà còn là nhắc tới những thực trạng trong ngành phần mềm, cũng như những nghiên cứu + kiến giải độc đáo của anh.

Blog [vinacode.net](#), có dịch khá nhiều bài viết từ coding horror sang tiếng Việt, những bạn lập trình viên không tự tin về ngoại ngữ của mình có thể vào và tìm đọc.

3. Signal vs Noise



The screenshot shows the homepage of Signal vs Noise. The title "About Signal v. Noise" is at the top center. Below it is a subtext: "We're the team that makes [Basecamp](#), everybody's favorite project management software. We wrote the best-selling books [REWORK](#) and [REMOTE](#). New here? [Start with our best hits](#)." A horizontal line labeled "BASECAMPERS" follows. Below this are two rows of circular profile pictures and names. The first row includes Jason Fried, David, Ryan, Sam, Jeff, Jeremy, and Jamie. The second row includes three more individuals whose profiles are partially visible.

Đây không phải là blog của 1 cá nhân, mà là blog của toàn bộ đội ngũ developer-designer-manager của cty 37signals. Cho những bạn nào chưa biết, đây là công ty tạo nên Basecamp – 1 phần mềm rất tốt hỗ trợ cho việc làm việc theo nhóm (Mình đã từng xài và công nhận nó tốt thật). 37signal còn nổi tiếng với 2 cuốn sách: Getting Real – Hướng dẫn cách viết 1 phần mềm thực sự, Remote – Cách xây dựng 1 start up. Cả 2 cuốn sách này đều rất hay và cuốn hút, các bạn nên đọc thử nếu có thời gian.

4. Jon Skeet (C# only)

Jon Skeet less info



bio	website	csharpindepth.com
	location	Reading, United Kingdom
	age	37
visits	member for	4 years, 10 months
	seen	3 hours ago
stats	profile views	607,075

592,602
reputation

• 206 • 3414 •
4855

Anh Jon Skeet này không được nổi tiếng bằng các blogger trên. Anh là người đã viết ra cuốn C# in Depth – 1 cuốn sách mô tả mọi ngóc ngách tường tận sâu xa về C#, là sách gối đầu giường của mọi senior developer C#.

Blog của Jon giải thích tường tận một số khái niệm, class trong C# mà ít người hiểu rõ. Anh còn là một thành viên rất tích cực của stackoverflow. Ngoài ra, anh còn khá vui tính, lần trước thấy email của Jon trên stackoverflow, mình gửi mail khen ngợi cuốn sách mà cũng được ảnh reply lại.

5. Simple Programmer

“I consider John to be one of the hardest working developers in the software business today” – Joe Colantonio

[in Share 0](#) [g+ Share 1](#) [Tweet 1](#) [f Share 0](#)

I'm **John Sonmez**.
I'm the founder of [Simple Programmer](#).
My passion is [taking the complex and making it simple](#).
I do this in a variety of different ways and I wear a lot of different hats.



Đây là blog mình xem hằng ngày, viết bởi John Sonmez. Anh chàng này đã đi làm developer một thời gian, sau đó chuyển sang dạy trên pluralsign, tư vấn, viết sách. Blog này đã thay đổi cách suy nghĩ của mình, rằng “lập trình viên cần biết nhiều điều hơn ngoài code”. Mình rất hâm mộ và rất muốn noi theo bước chân của John, đó cũng là một trong những lý do mình viết blog này.

Blog của John khá hay, bao quát mọi chủ đề: code, testing, phát triển sự nghiệp, xin việc,... Điểm nổi bật của blog là những bài viết về cách nâng cao giá trị của bản thân developer trên thị trường việc làm, cách làm bạn nổi bật giữa muôn vạn developer khác. Đạo này do ông vừa ra sách nên trên blog quảng cáo hơi nhiều, có thể bạn sẽ hơi khó chịu tí. Chút khó chịu đó sẽ không là gì so với những thứ bạn học được trên blog này.

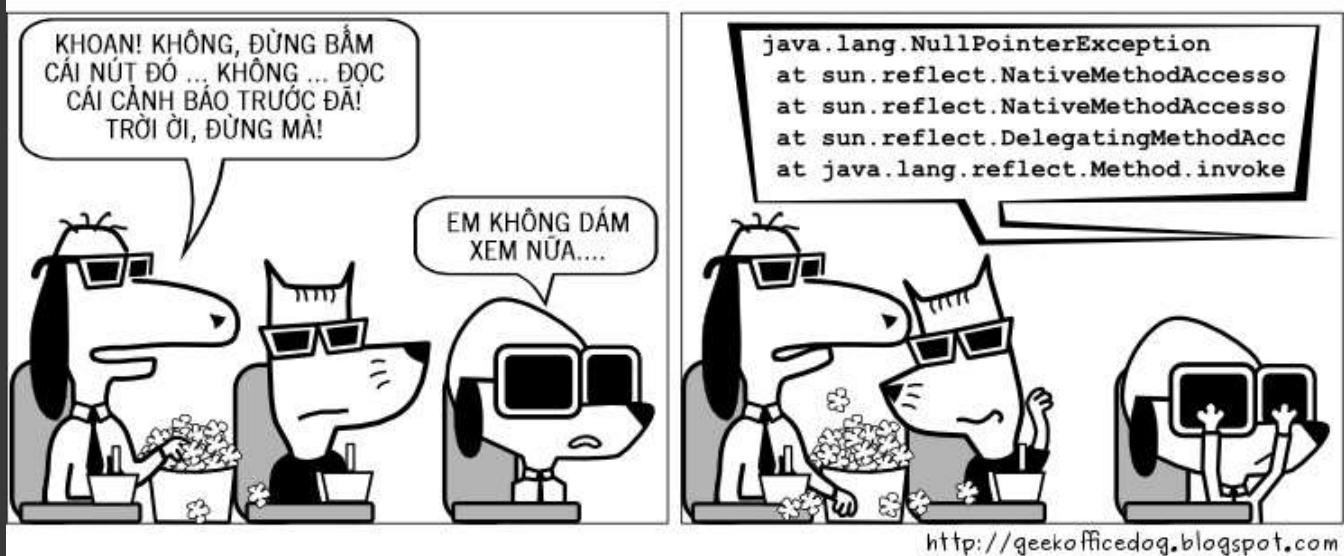
Một số mẫu comic hài hước về ngành IT – phần 1

Posted on 30/06/2015 by Phạm Huy Hoàng

Hôm nay hết ý tưởng để viết bài rồi, đành tìm vài cái comic khá vui về ngành IT, dịch sang tiếng Việt rồi chia sẻ với mọi người vậy :D.

Nỗi đón đau kinh hoàng của đời developer

XEM NGƯỜI DÙNG TEST TRÊN HỆ THỐNG, CẢM GIÁC GIỐNG NHƯ ĐANG XEM PHIM KINH DỊ 3D



Vấn đề thường gặp của lập trình viên – Khách hàng đòi sửa đổi requirement.



Khi cấp trên mù công nghệ mà hổ báo



Khi nhân viên tưởng cấp trên mù công nghệ nên hổ báo

Huckles

By Drake Emko & Jen Brodzik



<http://huckles.org>

Copyright © 2003 Drake Emko & Jen Brodzik

Khi bị bọn lập trình viên khác hổ báo thích thẻ hiện

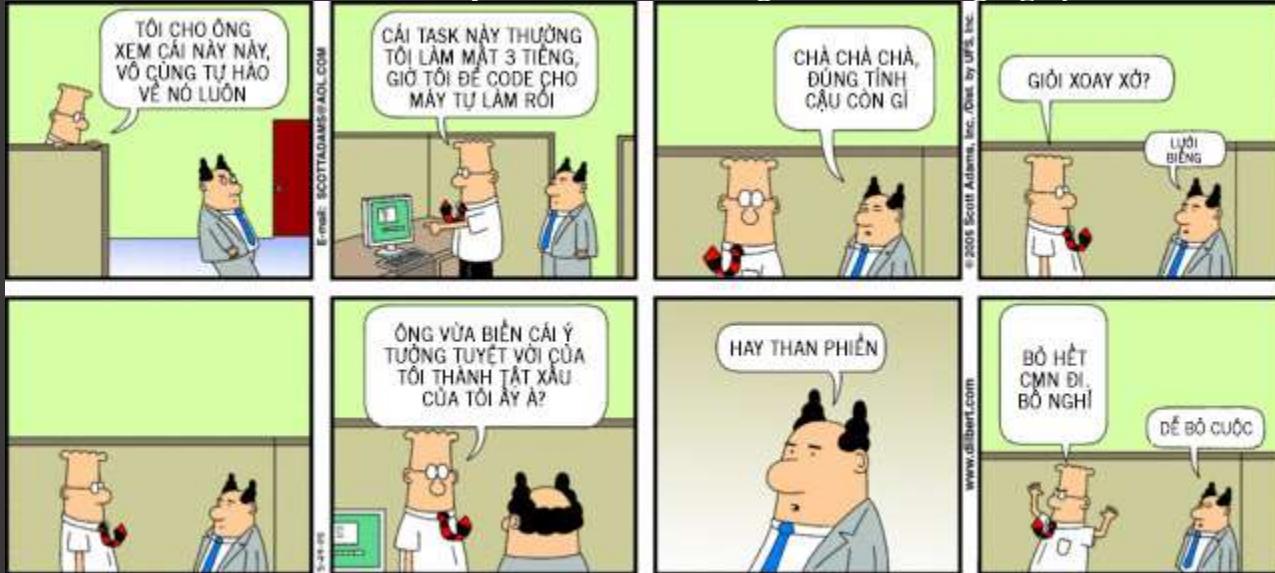


<http://geekofficedog.blogspot.com>

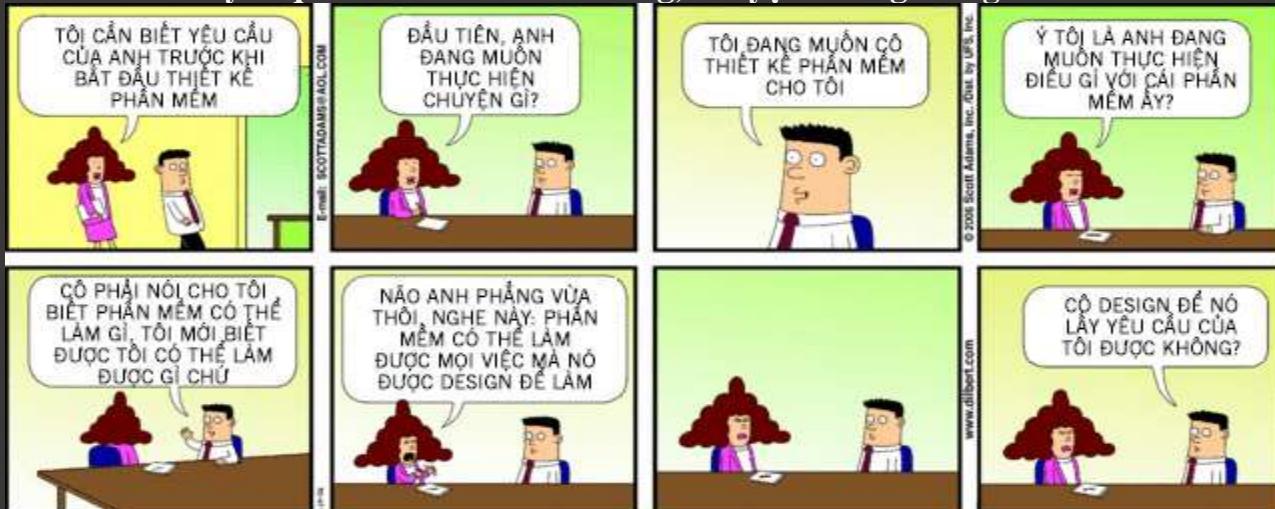
Nỗi niềm khi phải báo cáo với các bác cấp trên dở hơi, chụp mũ



Nỗi niềm khi có cấp trên bảo thủ cứng đầu, mù công nghệ



Lấy requirement từ khách hàng, chuyện không bao giờ cũ



Truyện còn khá nhiều, mời các bạn xem tiếp [phần 2](#).

Một số mẫu comic hài hước về ngành IT – phần 2

Posted on 02/07/2015 by Phạm Huy Hoàng

Có vẻ các bạn khá thích thể loại comic này, [phần 1](#) mình nhận được khá nhiều views. Do đó mình đăng tiếp phần 2.

Ngày xưa mỗi lần compile code phải đợi cả tiếng, mới dùng lý do này được



Bà mẹ lập trình viên “bá đạo” (Cái tên của con bà sẽ gây ra lỗi SQL Injection, drop toàn bộ bản Student, dẫn tới mất dữ liệu)



Nỗi khổ của lập trình viên C# (Bị xa lánh chắc tại vì 2 cái kia free, còn .NET có phí)



Chuyện quá quen thuộc với các lập trình viên



Đừng mừng vội khi được giao việc chịu trách nhiệm cho dự án



Hacker trên phim và đời thật – Sự bá đạo của các anh IT



Đôi khi ta cố gắng áp dụng công nghệ, mà không nghĩ ra cách đơn giản hơn để giải quyết vấn đề



Bonus tấm cuối, chào thân ái và quyết thắng.



Cách tiếp cận 1 ngôn ngữ/công nghệ mới – Phần 1

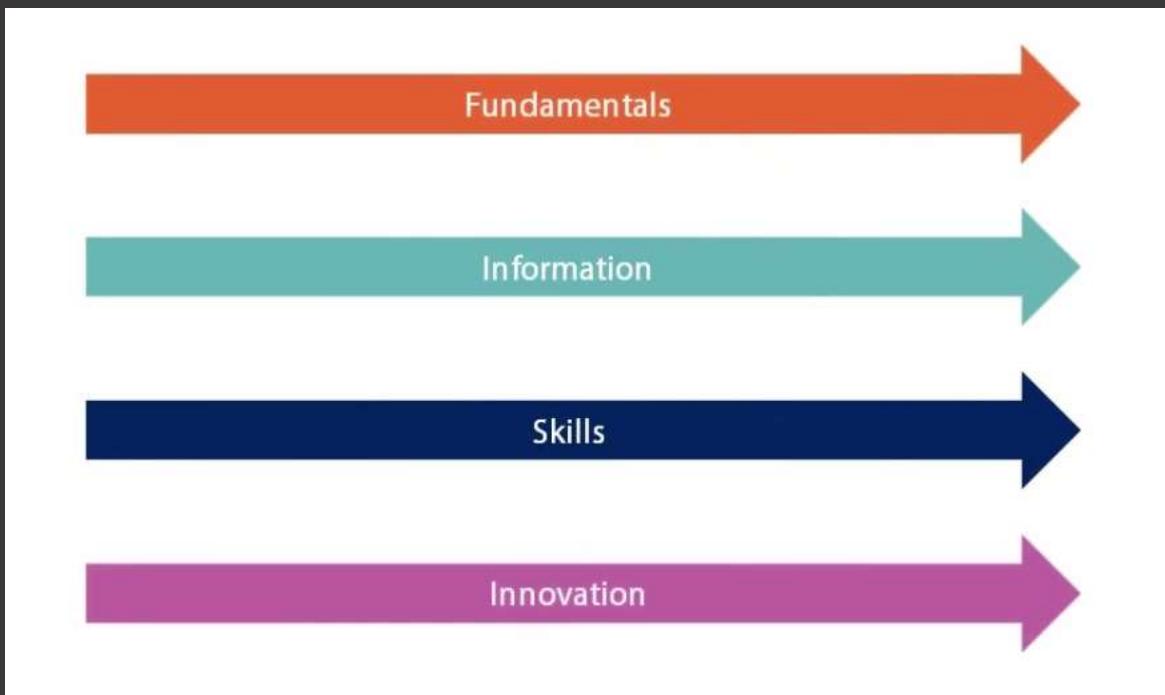
Posted on 23/06/2015 by Phạm Huy Hoàng

Mình đã từng nói về tầm quan trọng của việc cập nhật kiến thức ở [bài viết trước](#):
Không như các ngành khác, kiến thức trong ngành IT rất nhanh hết hạn. Với ngành xây dựng, xây một cây cầu cách đây 50 năm cũng chẳng khác gì xây một cây cầu bây giờ. Với ngành y, bệnh cảm cúm cách đây 50 năm triệu chứng cũng giống bệnh cảm cúm bây giờ. Nhưng với ngành IT, công nghệ, ngôn ngữ hoặc framework nổi tiếng cách năm 10-15 năm giờ chẳng ai xài nữa cả.

Như đã hứa, mình sẽ dành bài viết này để hướng dẫn các bạn cách tiếp cận một công nghệ mới. Đây là những cách mà mình tự tìm ra, tự tổng hợp trên mạng, cộng với một số lời khuyên của các bậc đàn anh. Bản thân mình thấy nó khá là hữu dụng, hi vọng chúng cũng sẽ hữu dụng với các bạn.

Nói về lý thuyết một chút, những kiến thức bạn cần học về một công nghệ có thể chia làm 4 loại sau (Lấy ngôn ngữ C# làm ví dụ):

- Nền tảng (Fundamentals)
- Kiến thức (Information)
- Kỹ năng (Skills)
- Nâng cao (Innovation)



1. Nền tảng (Fundamentals)

Đây là những kiến thức cơ sở nhất, là những viên gạch đặt nền móng cho kiến thức sau này (VD như: cấu trúc dữ liệu, OOP, vòng lặp, đệ qui, callback, 1 số mô hình MVC MVVM, cơ chế hoạt động của web, ...). Vì chúng là kiến thức nền tảng, mang tính học thuật nhiều nên đôi khi khá là phi thực tế và buồn ngủ. Chắc hẳn ai cũng từng nhức đầu đau não khi nghe các thầy giảng về sự kiện, con trỏ hàm, cây nhị phân, đệ qui....

Tuy nhiên, nếu nắm vững những kiến thức nền tảng này, bạn sẽ thấy việc chuyển đổi qua lại giữa các ngôn ngữ khác nhau rất dễ dàng, vì chúng được xây dựng dựa trên nền tảng chung (Như bản thân mình, vì đã rõ cơ chế get/post, giao tiếp giữa client/server, mô hình MVC, mình có thể học nhanh Zend của PHP, Struts2 của Java, ASP.MVC của C#).

Trường đại học chủ yếu dạy những kiến thức này, do đó đôi khi bạn sẽ thấy chương trình học khá khô khan. Hãy nhớ điều mình từng học khi xem phim kiếm hiệp thời xưa, **để học được võ công thượng thừa, phải rèn những chiêu thức cơ bản trước**. Những chiêu thức hoa mĩ đều từ cơ bản mà ra cả. Ngoài ra, những kiến thức cơ bản này thường “sống lâu”, rất khó “hết hạn” : hàm sort qua 10,20 năm vẫn giữ nguyên cách sort; cấu trúc dữ liệu stack, binary tree, mô hình MVC qua 10,20 năm vẫn không hề thay đổi.

2. Kiến thức (Information)

Đây là những kiến thức bậc cao hơn, liên quan tới từng ngôn ngữ/framework chuyên biệt (VD như LINQ, Event, WinForm, WebForm, ...). Những kiến thức này gắn liền với thực tế, có thể áp dụng được ngay vào làm việc. Để học nhanh, áp dụng được những kiến thức này, các bạn phải có fundamental vững. Mình từng gặp khó khăn khi viết ajax, viết jQuery, function lồng vào nhau v...v. Khi mình hiểu ra chúng gọi là callback, mình học và viết code ajax, jQuery dễ hơn nhiều.

Một số trường dạy nghề (APTech, Nhất Nghệ ...) thường tập trung nhiều vào kiến thức dạng information, lướt qua kiến thức cơ bản dạng fundamentals. Do đó học viên được đào tạo ra thường có kiến thức thực tiễn, có thể làm được việc ngay. Nhưng vẫn đề chung mà một số bạn hay gặp là: tuy làm được nhưng lại không hiểu cơ chế hoạt động, khi gặp lỗi ko biết nguyên nhân, không biết cách sửa. Lý do là vì kiến thức cơ bản (fundamentals) không đủ.

Một điều cần lưu ý nữa là những kiến thức dạng này khá nhanh “hết hạn”, vd như cách routing trong MVC 4 sẽ khác MVC 2, một số hàm trong Entity Framework 6 sẽ khác Entity Framework 4. Do đó nếu không kịp cập nhật, bạn sẽ dễ trở nên lỗi thời, vì kiến thức cũ không sử dụng được nữa.

3. Kỹ năng (Skills)

Đây là **loại kiến thức đáng giá nhất** (theo nghĩa đen), các công ty sẽ trả lương cho bạn nếu bạn có skills, có thể làm được việc. Kỹ năng có thể học được 1 phần từ trong sách vở, nhưng phần lớn bạn học được là do quá trình làm việc lâu dài, tiếp xúc nhiều với một công nghệ, giải quyết những tình huống cơ bản và phức tạp.

Ví dụ như: Infomation là việc bạn biết cơ chế routing, binding của ASP MVC. Skill là việc bạn biết áp dụng cơ chế routing, binding để tạo 1 trang search, insert, update. Skill phức tạp hơn là khi bạn đọc yêu cầu của khách hàng, bạn sẽ mường tượng ra cách viết front end thế nào, back end ra sao, bắt tay vào code ở đâu.

Lương ở các vị trí senior thường cao hơn, lý do là họ đã tiếp xúc với công nghệ nhiều, kỹ năng liên quan tới công nghệ đó sẽ giỏi hơn. Skill có dựa trên infomation, do đó nó cũng khá dễ hết hạn. Nếu bạn là senior ngôn ngữ Cobol, Basic nhưng thị trường không cần những skill đó nữa, skill của bạn sẽ trở nên vô dụng. Hãy tập trung đầu tư làm mới skill cho mình nhé.

4. Thuần thực (Innovation)

Đây là cảnh giới tối cao của kiến thức, đạt tới cảnh giới này bạn sẽ được gọi là **senior, master**, hoặc được phong **thánh**. Để đạt được cảnh giới này, ngoài quá trình làm việc, tiếp xúc lâu dài với công nghệ, họ còn phải bỏ thời gian đào sâu, mày mò, nghiên cứu công nghệ đó.

Ngoài những kiến thức chung, họ còn biết vô số những thứ chuyên sâu như: Code C# được biên dịch như thế nào, quan hệ giữa các component trong C#, performance của Interface và Abstract class, ... Bạn không cần lo lắng quá về cảnh giới này, bản thân chúng ta có thể lên được vị trí cao, có mức lương thoải mái mà không cần những kiến thức dạng innovation thế này.



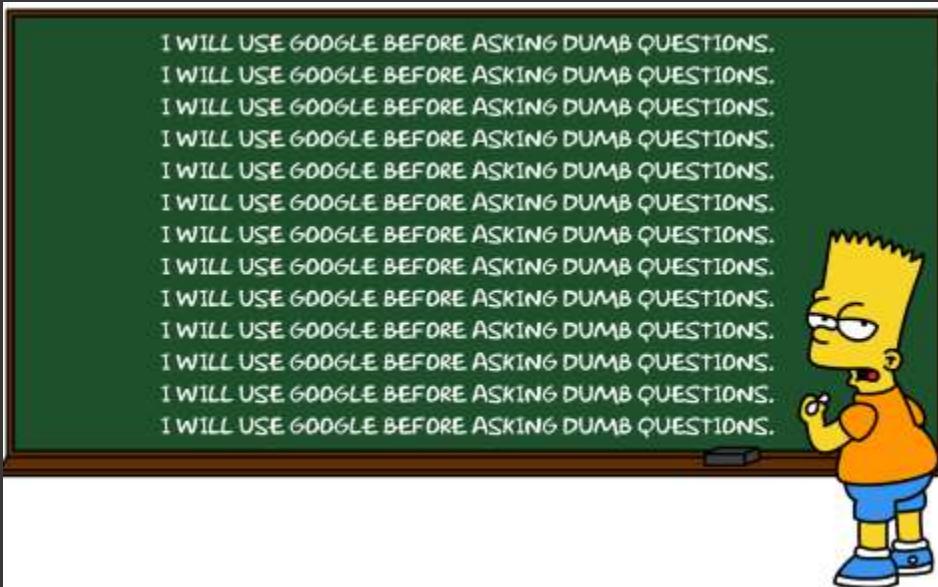
Ở phần này, mình đã có 1 cái nhìn tổng quát về những kiến thức bạn cần có khi tiếp xúc với một công nghệ mới. Ở [phần tiếp theo](#), mình sẽ hướng dẫn các bạn từng bước để dần đạt được những kiến thức này, chia sẻ những khó khăn mình từng gặp phải cũng như cách giải quyết. Mong các bạn đón đọc.

Cách tiếp cận 1 ngôn ngữ/công nghệ mới – Phần 2

Posted on 25/06/2015 by Phạm Huy Hoàng

Nối tiếp [phần 1](#), ở phần này mình sẽ nói rõ hơn về quá trình tiếp cận công nghệ của bản thân. Trước khi bắt đầu, mong các bạn hãy giữ 3 tư tưởng sau:

1. Học một ngôn ngữ/công nghệ mới **không khó**. Mình biết có nhiều bạn rất ngại, rất sợ học cái mới, hễ nghe nói cái gì là lạ là lắc đầu nguầy nguậy, bảo “không biết”. Chúng ta nên có tư tưởng là “không phải không biết mà là chưa biết, chịu khó tìm hiểu một tí là biết thôi thôi”. Mình đã giải thích lý do chúng ta có thể tiếp cận công nghệ mới một cách dễ dàng ở [bài viết này](#).
2. Để học được nhiều cái mới, bạn cần phải **giỏi tiếng Anh, không ngại đọc**(Không cần giỏi cả 4 kỹ năng, chỉ cần giỏi reading là được). Ngoại trừ một số ngôn ngữ cũ như C, C++ được nhiều dạy ở nhiều trường , có tài liệu tiếng Việt, các công nghệ mới như NodeJS, AngularJS, Entity Framework thường chỉ có tài liệu hoặc hướng dẫn tiếng Anh. Nếu chỉ chăm chăm tìm tài liệu tiếng Việt, chỉ biết há miệng chờ hàng người ta dịch sẵn, bạn sẽ đi sau thời đại. Ngoài ra, với vốn tiếng Anh kha khá, khi có bug hoặc gặp vấn đề khó giải quyết, bạn sẽ dễ google và tìm câu trả lời hơn.
3. Hạn chế hỏi linh tinh,hãy **google trước khi hỏi** . Mình rất đồng tình với quan điểm “không biết phải hỏi, không giấu dốt”. Tuy nhiên, dân lập trình viên nói chung rất ghét những câu hỏi ngu, lười suy nghĩ. Trước khi hỏi, hãy thử tìm google trước. Có khi bạn hỏi chỉ mất 1 phút là có câu trả lời, google để tìm câu trả lời mất tới 1 tiếng. Nhưng trong 1 tiếng đó, bạn sẽ học được rất nhiều điều liên quan khác, cả những điều bạn không biết mình cần phải hỏi.



Các bạn nhớ đọc lại [phần 1](#) để hiểu khái niệm về: fundamental, information và skill nhé. Dưới đây, mình sẽ chia sẻ quy trình mình áp dụng để học một công nghệ mới. Tất nhiên không phải công nghệ nào cũng vậy, tùy vào việc mình rành kiến thức nền tảng – fundamental của công nghệ đó tới đâu:

Tìm tài liệu học – Giai đoạn sơ khởi

Đây là **bước quan trọng nhất**. Nếu có người quen rành công nghệ này, bạn có thể nhờ họ chỉ từ khóa, tên sách, website v...v để mình có thể tự tìm hiểu. Họ cũng sẽ chỉ cho những gì cần học. (VD: mình muốn học thiết kế web, cần học trước về html, css, javascript, jQuery.).

Trường hợp xui xẻo hơn, nếu bạn không có người quen, có thể lên amazon.com, đánh tên công nghệ mình muốn học, sau đó chọn 1,2 cuốn ebook đứng đầu, tìm bản ebook và bắt đầu học. Lưu ý là **chỉ cần 1,2 cuốn, không nhiều hơn nhé**, nhiều hơn bạn sẽ lười và không đọc đâu. Minh biết có nhiều bạn tải gần cả trăm MB ebook, tưởng mình giỏi, nhiều tài liệu mà bản thân họ chẳng đọc bao giờ.

Bắt đầu học – Giai đoạn nhập môn

Tại sao mình lại khuyến khích bạn học từ sách, mà không phải là học qua website, forum... hay gì đó. Lý do là khi xuất bản một cuốn sách, tác giả thường trau chuốt + soạn sẵn một chương trình học cho bạn. Các kiến thức được trình bày trong sách theo thứ tự tuần tự mạch lạc. Đây là giai đoạn các bạn bổ sung kiến thức dạng nền tảng (fundamentals) và information về công nghệ mình muốn học.

Sách technical dĩ nhiên vẫn có code. Mình khuyên các bạn nên làm theo hướng dẫn trong sách, setup môi trường, **phải code theo** (Đừng đọc code rồi gật gù ở ở nhé, chẳng thấm được gì đâu). Lưu ý là **gõ code bằng tay** để hiểu, không **copy code vào rồi chạy** nhé. Vừa gõ, vừa sửa code, vừa thử nghiệm, các bạn sẽ dần dần có một cái nhìn rõ ràng hơn về công nghệ mình đang học. Mình đã tự học HTML, CSS, Javascript, jQuery theo cách này, thông qua series sách Head First.

Một số series hay để nhập môn: **xxx For Dummies**, **Head First xxx**, **sách khác của O'Reilly**. Những sách này có ngôn ngữ đơn giản, ảnh minh họa nhiều, việc học rất vui và nhẹ nhàng. Có một số cách học khác như học qua video trên [pluralsight](#), code ngay với [codeschool](#), cũng khá trực quan và dễ hiểu.



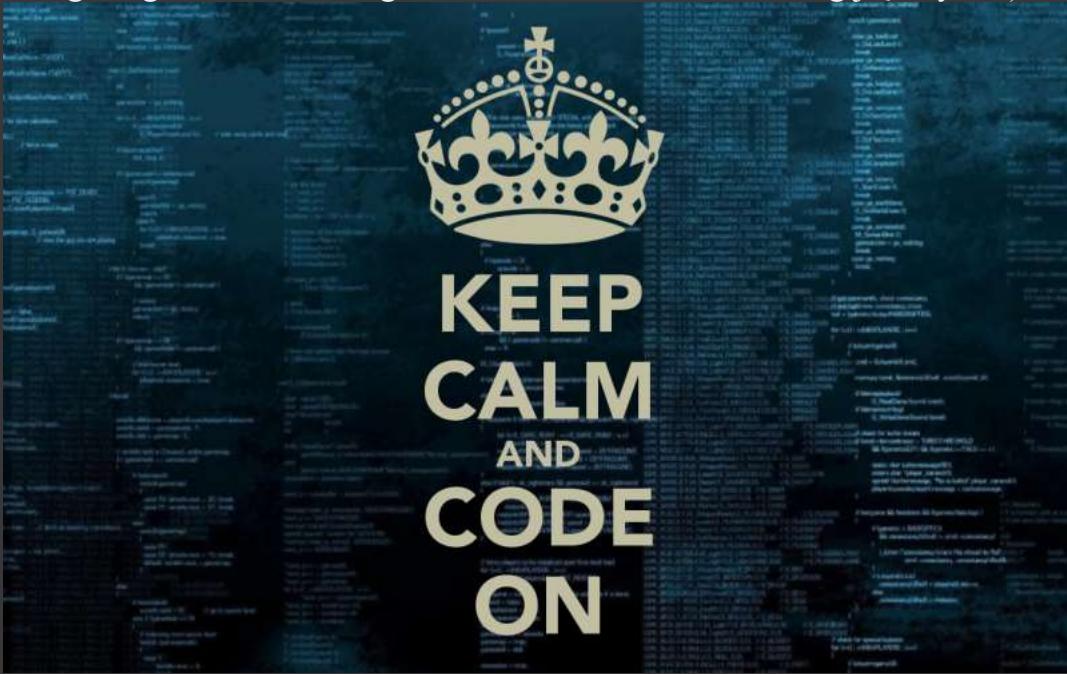
Áp dụng kiến thức – Giai đoạn nâng cao

Bạn **đừng lo nếu mình đọc được 1 nửa cuốn sách đã chán, mình cũng từng như vậy**. Đọc được 1/2 hoặc 2/3 cuốn sách, bạn đã có kha khá kiến thức nền tảng, cùng 1 chút kiến thức chuyên sâu về công nghệ mình học. Lúc này sách vở không còn nhiều tác dụng nữa, bạn hãy thử áp dụng kiến thức mình đã học vào thực tế.

Bằng cách nào? **Hãy thử viết một phần mềm to-do-list, hoặc một trang web to-do-list, hoặc 1 blog bằng công nghệ mình đang học**. Đây là những phần mềm dễ viết, yêu cầu rõ ràng, nhưng lại cho bạn cơ hội để tự trải nghiệm công nghệ mình học qua việc làm các chức năng: Chức năng hiển thị, thêm bớt xóa sửa, kết nối database ,.... Nếu gặp khó khăn trong quá trình làm, bạn có thể xem lại sách, tìm cách giải quyết tương tự, hoặc lên [stackoverflow](#) để hỏi. Sau khi làm xong, hãy róng trau chuốt source code, sau đó upload nó lên github. Bạn vừa có project để tham khảo, hướng dẫn cho người sau, vừa có project để giới thiệu cho nhà tuyển dụng.

Ở giai đoạn áp dụng này, bạn sẽ học được nhiều skill, qua đó cũng cố thêm fundamental và information. Bạn tốt của bạn ở giai đoạn này là [stackoverflow](#) hoặc 1 số sách dạng cookbook. Những sách cookbook này khá hay, chúng hướng dẫn cách

dùng công nghệ để giải quyết một số yêu cầu thường gặp khi code (Cách parse 1 string sang DateTime trong C#, cách validate 1 form trong jQuery, ...)



Trên đây là 3 giai đoạn các bạn sẽ trải qua trong quá trình tiếp cận công nghệ. Khi cảm thấy mình có thể giải quyết 80% mọi vấn đề gặp phải, các bạn đã đạt tới cuối giai đoạn “áp dụng”. Các bạn có thể bỏ thời gian nghiên cứu chuyên sâu hơn, hoặc chuyển qua tìm hiểu công nghệ mới, tùy vào đam mê của mỗi người.

Đĩ nhiên mỗi người có một cách học khác nhau, đây chỉ là cách tiếp cận của cá nhân mình. Các bạn có thể góp ý, đóng góp cách học của mình bằng cách comment cho bài viết. Hoan nghênh mọi ý kiến đóng góp của các bạn. Xin chia sẻ một course pluralsight khá hay về cách học: <http://www.pluralsight.com/courses/learning-technology-information-age>

Con đường phát triển sự nghiệp (Career path) cho developer

Posted on 18/06/2015 by Phạm Huy Hoàng

Các bạn sinh viên còn đang học hoặc mới ra trường sẽ khó hình dung được về những vị trí, chức danh trong ngành lập trình. Mình viết bài này nhằm giải đáp một số thắc mắc các bạn thường hỏi như:

- Mới đi làm em có chức danh gì, công việc thế nào.
- Code lâu thì lên được chức gì, lương cao không?
- Em thích code thôi, không thích làm trưởng nhóm, em nên định hướng thế nào.

Hiểu rõ con đường nghề nghiệp của ngành developer, các bạn sẽ dễ định hình phát triển tương lai của bản thân, cũng như dồn sức vào con đường mình đã chọn.

Mình chỉ liệt kê con đường nghề nghiệp của 1 developer, vì bản thân mình cũng là developer. Con đường của 1 tester (QA engineer) cũng có 1 số chức danh tương tự, nhưng lên cao sẽ khác. Các chức vụ sẽ được mình miêu tả theo thứ tự từ thấp lên cao nhé.



Fresher/Junior Developer

Các bạn sinh viên đi thực tập hoặc mới ra trường thường được chức danh này. Số năm kinh nghiệm của Junior Developer thường vào khoảng 6 tháng – 1 năm. Mức lương thì tùy vào khả năng của bạn, thường là từ 300-500\$.

Do chưa có kinh nghiệm, fresher/junior thường **được các công ty đào tạo lại**, do đó khi phỏng vấn fresher các công ty thường chỉ xét khả năng suy nghĩ logic, khả năng

lập trình, tiềm năng lập trình của bạn. Cá nhân mình thấy chương trình đào tạo Fresher của FSOFT cũng khá tốt, có dạy nhiều thứ mà ban sẽ tiếp xúc khi làm việc (mặc dù lương fresher hơi thấp).

Do chưa có kinh nghiệm, mọi người thường không đòi hỏi ở bạn quá cao. Công việc của 1 junior thường là tìm hiểu project hiện tại, code các module nhỏ, đơn giản, fix bugs, có thể có sự trợ giúp/review của senior. Ở giai đoạn junior, các bạn hãy cố gắng tranh thủ học code, học cách thức làm việc, học hỏi kinh nghiệm của các bậc senior đi trước.



Developer

Code được 1 thời gian, khoảng 1-3 năm, các bạn sẽ được gọi là Developer (Nhiều bác lên thẳng Team Leader hoặc Senior, tùy công ty). Ở giai đoạn này, bạn đã làm qua một số project, khá rành về 1 số công nghệ. Mức lương của developer vào khoảng 600-900\$ tùy vào công ty.

Màn phỏng vấn cho developer thường khó hơn. Người phỏng vấn sẽ hỏi bạn về những project đã làm, các khó khăn bạn đã gặp phải, cách giải quyết? Ngoài ra, buổi phỏng vấn sẽ tập trung vào những công nghệ bạn đã ghi trong CV. Vì developer đã có kinh nghiệm, các bạn sẽ không còn “được” các anh senior kèm cặp, và cũng khó mà lấy danh nghĩa junior để hỏi, nhò vả hay mắc lỗi nữa.

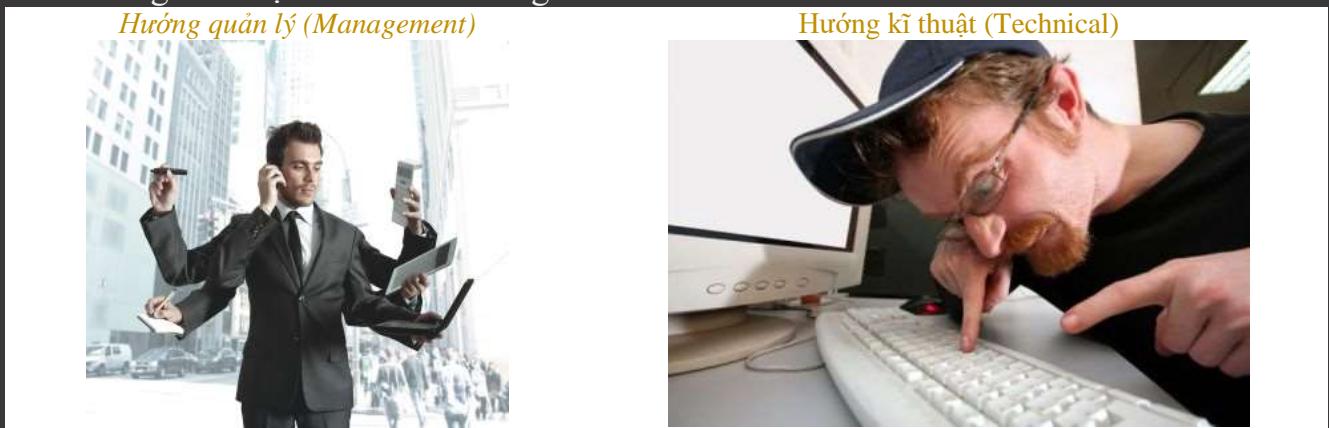
Ở giai đoạn này, bạn đã được code một số module phức tạp hơn, tham gia meeting, code review, thảo luận với khách hàng v...v. Đây là giai đoạn để bạn dồn nén kiến thức, kinh nghiệm, gây dựng danh tiếng để lên nấc tiếp theo trong bậc thang nghề nghiệp.

Lý thuyết là vậy nhưng thực tế, thuở làm FSOFT mình ăn lương junior mà làm công việc của developer, việc khó lâu lâu cũng đùn mình luôn, nhò vạy mình học hỏi cũng được khá nhiều.



Quản lý hay kĩ thuật?

Ở giai đoạn sau, bạn đã có thể xác định con đường cho mình. Nếu muốn tập trung vào code và kĩ thuật, bạn có thể đi theo hướng technical: **Senior Developer => Technical Lead => Software Architecture**. Nếu muốn làm việc với quy trình và con người, bạn nên đi theo hướng quản lý: **Team Lead => Project Manager => Manager**. Ở giai đoạn đầu, lằn ranh giữa 2 con đường này khá mờ nhạt, nhưng càng lên cao lại càng trở nên rõ ràng. Các bạn có thể xem bảng tóm tắt sau:



Team Leader

Bạn trở thành leader của 1 team nhỏ nhở, khoảng 3-6 thành viên. Ngoại trừ code ra, bạn còn phải họp hành với cấp trên, báo cáo với khách hàng, quản lý cấp dưới. Ở giai đoạn này, bạn sẽ dần học thêm 1 số kĩ năng lãnh đạo, kĩ năng quản lý v...v. Ở 1 số cty nhỏ, developer lâu năm, có kinh nghiệm sẽ lên team leader. Bạn vẫn còn khá nhiều thời gian code,

Senior Developer

Sau một thời gian làm việc, bạn nắm vững, hiểu sâu và rộng nhiều công nghệ + qui trình. Ở vị trí này, ngoại trừ khả năng code “thần thánh”, bạn còn phải biết đưa ra design và solution. Ngoài ra, bạn còn phải hướng dẫn chỉ bảo các em junior mới vào, cũng như tham gia code review v...v. Đôi khi senior dev cũng kiêm luôn vị trí team leader, do đó bạn cũng cần một chút kĩ năng diễn đạt và lãnh đạo. Mức lương cho Senior Developer cũng khoảng 1000-1500\$(hoặc hơn)

*code giỏi có thể sẽ làm thành viên trong team
tôn trọng hơn. Mức lương cho team lead
thường khoảng 1000-1500\$*

Project Manager

*Lên đến vị trí này, bạn sẽ có rất ít hoặc hầu
như không có thời gian code. Đa phần thời
gian của bạn dùng để đọc báo, lướt voz, lướt
webtretho Điều này, công việc chính của
ban bây giờ là báo cáo, quản lý, lãnh đạo, lâu
lâu bạn còn bị bắt đi phỏng vấn 1 số ứng viên
để tuyển thành viên cho dự án nữa. Bạn là
người quyết định sự thành bại của 1 dự án, do
đó nếu dự án thành công bạn sẽ được thưởng 1
khoản bonus kha khá, tùy ctv. Mức lương cho
PM vào khoảng 1000-2000\$*

Manager/Director

*Chúc mừng, ở vị trí này bạn đã được gọi là
sếp, cấp trên, lãnh đạo, ... Lúc này bạn sẽ
không có thời gian mà code, suốt ngày họp
hành, giao việc, phỏng vấn, trao đổi với các bộ
phận, phòng ban, xử lý việc hành chính...
Mình không có thông tin về mức lương, thường
v...v của vị trí này.*

Technical Lead

Bạn cần hiểu biết về công nghệ sâu và rộng, vì chính bạn là người lựa chọn công nghệ, qui trình... cho 1 dự án. Những quyết định lớn về thiết kế, cấu trúc code ... sẽ do bạn chịu trách nhiệm. Ở giai đoạn này, ngoài việc technical “cứng”, bạn còn phải giỏi thuyết trình, hướng dẫn, giải thích... Vì sao á? Lead đưa ra vấn đề thì phải giải thích hợp lý, thành viên khác nó mới hiểu, nè và làm theo chứ. Mức lương cho vị trí này vào khoảng 1500-2500\$.

Software Architect

Muốn đạt chức danh này, ít nhất bạn phải có 10-20 năm trong ngành. (Nhìn thẳng nào mặt mũi trẻ măng mà vỗ ngực tự xung SA thì đừng tin). Công việc của bạn khá gian khổ: Từ một yêu cầu “mơ hồ” của khách hàng, bạn phải làm việc với BA để đánh giá solution, làm việc với PM để xây dựng 1 team, làm việc với Technical Lead để thiết kế, đưa ra các quyết định quan trọng về kiến trúc. Vị trí này mặc dù không có quyền quản lý, nhưng lại có khá nhiều quyền lực ngầm. Mức lương cũng ngang ngửa hoặc cao hơn cả manager.

Ngoài những con đường trên, các bạn có thể đi theo hướng Sales, Kỹ sư cầu nối (BrSE), Business Analyst (Bên itviec có 1 bài viết khá hay về BA ở đây: <http://blog.itviec.com/2015/03/business-analyst/>). Trong phạm vi bài viết, mình không cover hết được toàn bộ những bước phát triển trong ngành này, mong nhận được sự bổ sung, góp ý của các bạn.

Những điều trường đại học không dạy bạn – Phần 1

Posted on 04/06/2015 by Phạm Huy Hoàng

Lang thang trên mạng đọc được một bài viết có tên là “Những điều trường học không dạy bạn”, bỗng dung muôn viết một bài tương tự, dành cho dân developer tại mình. Kinh nghiệm làm việc của mình cũng kha khá, có thể sẽ không đầy đủ, rất mong comment góp ý ủng hộ của các bạn. Có rất nhiều điều chúng ta không được học ở trường (Nhưng lại vô cùng cần thiết), mình tạm chia ra làm 3 phần.

Đây là phần đầu trong series bài viết “Những điều trường đại học không dạy bạn”:

1. **Kĩ thuật lập trình**
2. Cách nâng cao giá trị bản thân
3. Thành công và thăng tiến trong môi trường làm việc

Ở phần 1 này, mình sẽ nói về **kĩ thuật lập trình**.

Ở Việt Nam, đa phần các bạn lập trình viên thường là sinh viên tốt nghiệp ngành Khoa học Máy tính (Computer Science) ở trường đại học, 1 số bạn tự học hoặc học qua 1 số trung tâm. Mảng Khoa học Máy tính thường **nặng về tính khoa học, nghiên cứu**. Những kiến thức về hệ điều hành, thuật toán và cấu trúc dữ liệu v...v mà trường dạy là vô cùng cần thiết với các developer, mình không phủ nhận. Tuy nhiên, code, ngôn ngữ lập trình và design lại khá bị xem nhẹ. Do đó, khi bắt đầu làm việc, đa phần các bạn sẽ thiếu những kĩ năng sau đây:

Cách đọc và viết code

Khi còn ở đại học, khi bạn viết ra code chạy đúng, chạy được, giải quyết xong bài toán tức là bạn giỏi. Trong các kì thi cũng thế. Trong công việc thì khác, **chạy đúng** là yêu cầu bắt buộc, nhưng code được viết ra còn phải dễ hiểu, dễ đọc, dễ bảo trì và sửa chữa. Vì sao? Trong ngành này, code không phải chỉ viết 1 lần rồi bỏ đó, ta phải bảo trì, nâng cấp, sửa chữa thường xuyên. Như mình đã nói ở các bài viết trước, hãy chọn cách đơn giản, dễ hiểu, đừng chọn cách thông minh để rồi không ai hiểu.

Ngày xưa mình cũng từng là sinh viên giỏi, từng nghĩ mình code hay này nọ... sau khi đi làm hơn 1 năm rồi, đọc lại đóng **bùi nhùi trong như là code** của mình mới nhận ra ngày xưa mình trẻ trâu thế nào. Trường đại học dạy ta vô số thứ: lập trình hướng đối tượng, tính bao đóng, tính kế thừa v...v, nhưng chẳng ai dạy các bạn về SOLID – Những điều lập trình viên nào cũng cần nắm rõ; không ai dạy các bạn cách đặt tên hàm, tên biến, cách viết API cho dễ sử dụng; không ai dạy design pattern – một thứ để phân loại lập trình viên junior và senior. Các bạn hãy đọc 2

cuốn: [Code Complete](#) và [Clean Code](#), chỉ cần hiểu và áp dụng khoảng 30-50% những điều trong sách này, các bạn đã giỏi hơn 50% số developer còn lại rồi đấy



```
if (top != self) {
    function calcWidth() {
        var wW = 0;
        if (typeof window.innerWidth == 'number') {
            wW = window.innerWidth;
        } else if (document.documentElement.clientWidth > 0) {
            wW = document.documentElement.clientWidth;
        } else if (document.body.clientWidth > 0) {
            wW = document.body.clientWidth;
        }
        if (sH = document.documentElement.scrollHeight) {
            var wH = window.innerHeight || document.documentElement.clientHeight;
            if (sW = !document.all && (sH > wH)) {
                sH = wH;
            }
        }
    }
}
```

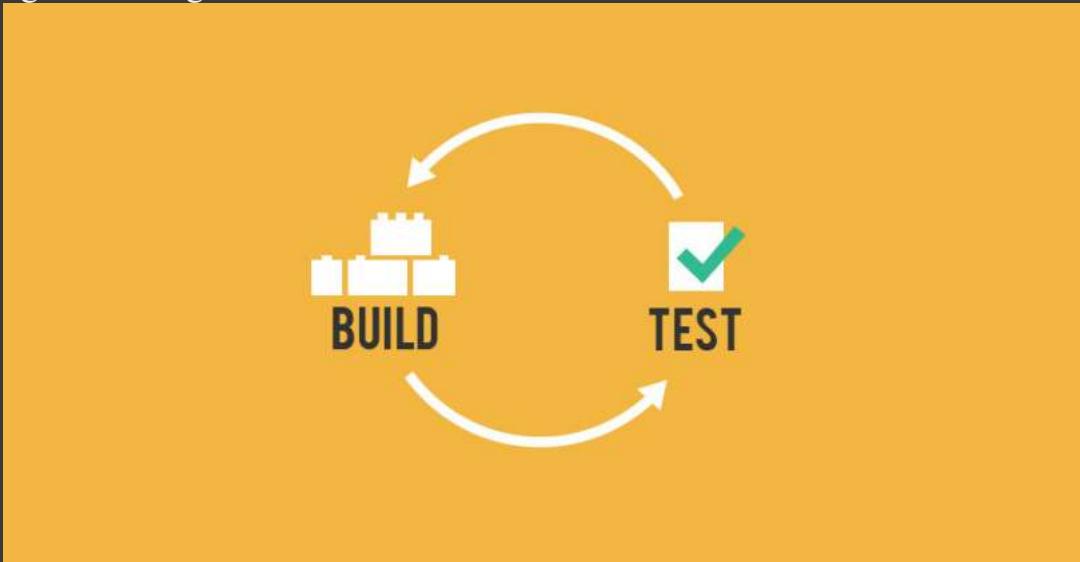
Sử dụng IDE, debug

Mình không vơ đưa cả năm, nhưng một số trường vẫn còn sử dụng phương thức **kiểm tra giấy** cho các kì thi lập trình. Sau đó giáo viên sẽ đọc code của từng học sinh, thật là cực cả thầy lẫn trò. Sinh viên code C, C++ trên notepad, hoặc 1 số ide, sau đó thì chạy code. Đến khi đi làm, nhiều bạn không biết sử dụng IDE như Eclipse, [Visual Studio](#)... không biết dùng Nuget, Maven .. Mình thì may mắn hơn, thời gian code chiếm 70% thời gian học, được sử dụng VS khi học C++, dùng NetBeans khi học Java, cơ mà cũng tới năm 3 mới bắt đầu biết set breakpoint để debug chương trình. Do đó mình thấu hiểu nỗi khổ của nhiều bạn học lập trình theo kiểu học thuật, **không được code đủ**.



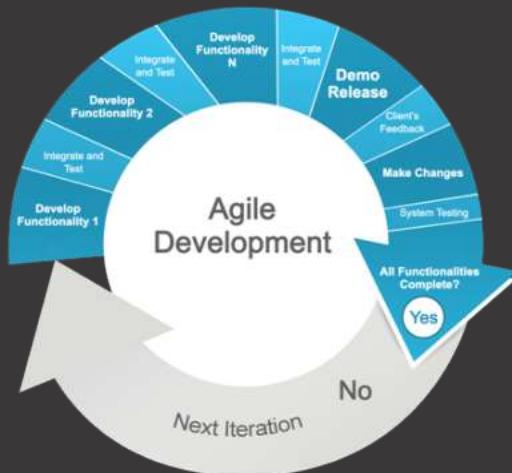
Testing, unit test

Trong chương trình học của 1 số trường vẫn có môn **Kiểm thử phần mềm**. Tuy nhiên, nhiều sinh viên CS vẫn ngáo ngơ không biết test case là gì, thế nào là black-box, white-box testing. Một số câu khoai hơn như: NUnit, JUnit, [Jasmine](#) là gì ... làm sao sử dụng mock, stub, dùng IoC càng không ai biết. Có người sẽ chu môi: Dào, tao đi code chứ có phải đi làm test đâu. Để làm một developer giỏi, phải chắc rằng code mình viết ra không lỗi. Để đảm bảo code không lỗi, phải có suy nghĩ của một tester, nghĩ ra những case để kiểm thử nó.



Agile Development

Ở trường đại học, chúng ta được học về “quy trình phát triển phần mềm”, học về waterfall, agile v...v (Một số trường không có). Tuy nhiên, chúng chỉ là những kiến thức nhảm chán trên giấy mà ai cũng quên ngay sau khi thi. Đến khi bắt đầu làm việc, bạn sẽ ngáo ngơ khi vào daily meeting, planning meeting, ko rõ quy trình ... vì không biết Scrum, XP là cái khỉ gì (Hồi vào FSOFT mình cũng ngáo ngơ, phải lên scrumtraining để học thêm).



Source code control system

Đây là một thứ khá đơn giản nên nhà trường cho rằng các bạn có thể tự học được. Hãy nhìn cách các nhóm SV năm nhất, năm 2 khổ sở làm bài tập lập trình nhóm: Mỗi người làm một phần, sau đó họp cả team ghép code lại, mất code là mất luôn (Mình cũng từng trải qua cảnh ấy, cũng may về sau đỡ hơn). Hậu quả là các SV mới ra trường phải được training lại về cách dùng SVN, dùng Git, hoặc TFS (Quảng cáo tí, chương trình Fresher của Fsoft có training cái này). Cuộc sống SV của bạn sẽ dễ thở hơn nếu bạn tự trang bị kiến thức về cách dùng **Git**, **SVN** cho mình.



Cách dùng thư viện và framework

Do bản chất của chuyên ngành Computer Science, các trường chỉ dạy 1 số ngôn ngữ như C++, Java để dạy các môn còn lại. Nhiều sinh viên ra trường vẫn không biết dựng 1 trang web như thế nào, ngôn ngữ này có framework gì hay, làm sao để hiểu và sử dụng API của 1 thư viện nào đó. Các trường chỉ dạy 1 vài mô hình MVC, MVP, MVVP trên giấy, còn cách dùng những thư viện, framework nổi tiếng như: Struts 2, ASP MVC, Ruby on Rails, jQuery ... còn tùy vào khả năng tự học của sinh viên. (Bài viết này không nói tới vài bạn có khả năng tự học giỏi, tự tìm hiểu và đã rành rọt vài ba framework khi mới ra trường nhé).



Hẹn gặp lại các bạn ở [phân 2](#), mình sẽ nói tới 1 điều nữa quan trọng không thua gì kĩ thuật lập trình: Cách nâng cao giá trị bản thân.

Những điều trường đại học không dạy bạn – Phần 2

Posted on [09/06/2015](#) by [Phạm Huy Hoàng](#)

Đây là phần hai trong series bài viết 3 phần “Những điều trường đại học không dạy bạn”:

1. [Kĩ thuật lập trình](#)
2. [Cách nâng cao giá trị bản thân](#)
3. [Thành công và thăng tiến trong môi trường làm việc](#)

Cảm ơn sự quan tâm các bạn đã dành cho [phân 1](#) của bài viết này. Nối tiếp [phân 1](#), ở phần này mình sẽ nói chi tiết cách **nâng cao giá trị bản thân** – một thứ **quan trọng không kém khả năng kĩ thuật**.

Hơn 90% các bạn sinh viên IT mới ra trường đều muốn có 1 công việc ổn định, lương cao, thoải mái (1 số bạn muốn làm-một-cái-gì-to-tát hoặc khởi nghiệp, mở công ty riêng mình không bàn tới, mình không có kinh nghiệm trong mặt này nên ko phán lung tung). Bạn cần biết một điều: Một công ty **chỉ trả lương cao cho bạn khi họ thấy giá trị của bạn cao, xứng đáng với mức lương họ bỏ ra**. Nếu bạn muốn thoải

mái lựa chọn công việc, mức lương đầy đủ, bạn phải tự tìm cách để nâng giá bản thân lên trong mắt họ. Dưới đây là những cách nâng cao giá trị bản thân mà trường học đã không dạy bạn:

Tìm hiểu thị trường

Mấy bạn IT chắc không học về “Tài chính vi mô”, nhưng chắc ai cũng biết khái niệm cơ bản về quy luật cung-cầu. Hãy xem sức lao động của bạn như 1 loại hàng hóa. Khi cầu thừa, cung thiếu thì giá hàng hóa tăng. Khi cầu thiếu, cung thừa thì giá hàng hóa giảm. Nói đơn giản: Lương cho developer Ruby on Rails ở VN khá cao, vì cầu khá cao nhưng cung chưa đạt tới. Lương PHP có thể hơi thấp vì số lượng developer PHP quá nhiều.

Mình đã từng nhắc sơ về chuyện này ở series: [Học ngôn ngữ lập trình nào bây giờ?](#). Các bạn sinh viên, nếu vẫn chưa xác định được hướng đi cho mình, hãy chịu khó tìm kiếm thông tin ở các trang ITViec, Linkedin, xem mức lương của các ngôn ngữ là bao nhiêu, ngôn ngữ lập trình nào đang cần nhiều ... sau đó tự trang bị mình kiến thức về ngôn ngữ, framework đó. Tìm hiểu rõ về thị trường cũng sẽ giúp bạn đỡ hối khi thỏa thuận lương lúc xin việc (Ngày xưa mình bị hối, vào FSOFT trả lương gross có hơn 7 củ T_T).



Mở rộng quan hệ

Quan hệ là thứ không thể thiếu trong bất kì ngành nghề nào. Mình biết được công việc hiện tại là nhờ thẳng bạn thân. Càng leo lên cao, bạn sẽ càng thấy tầm quan trọng của

quan hệ. Quen bạn bè ở nhiều công ty sẽ giúp bạn dễ đánh giá mức lương, dễ nhảy việc, và nhất là **dễ kiếm tiền**. Các công ty IT hiện nay đều có chính sách reference, chỉ cần giới thiệu được 1 người bạn vào công ty, các bạn có thể dễ dàng bỏ túi 4-10 triệu.

Làm sao để mở rộng quan hệ? Quan hệ có sẵn là bạn bè thân quen khi còn học đại học. Tới lúc đi làm, bạn sẽ quen biết nhiều người hơn. Một số mạng xã hội như facebook, linkedin cũng khá hữu ích. Viết blog như mình cũng là một cách để mở rộng quan hệ đấy.



Đầu tư vào bản thân

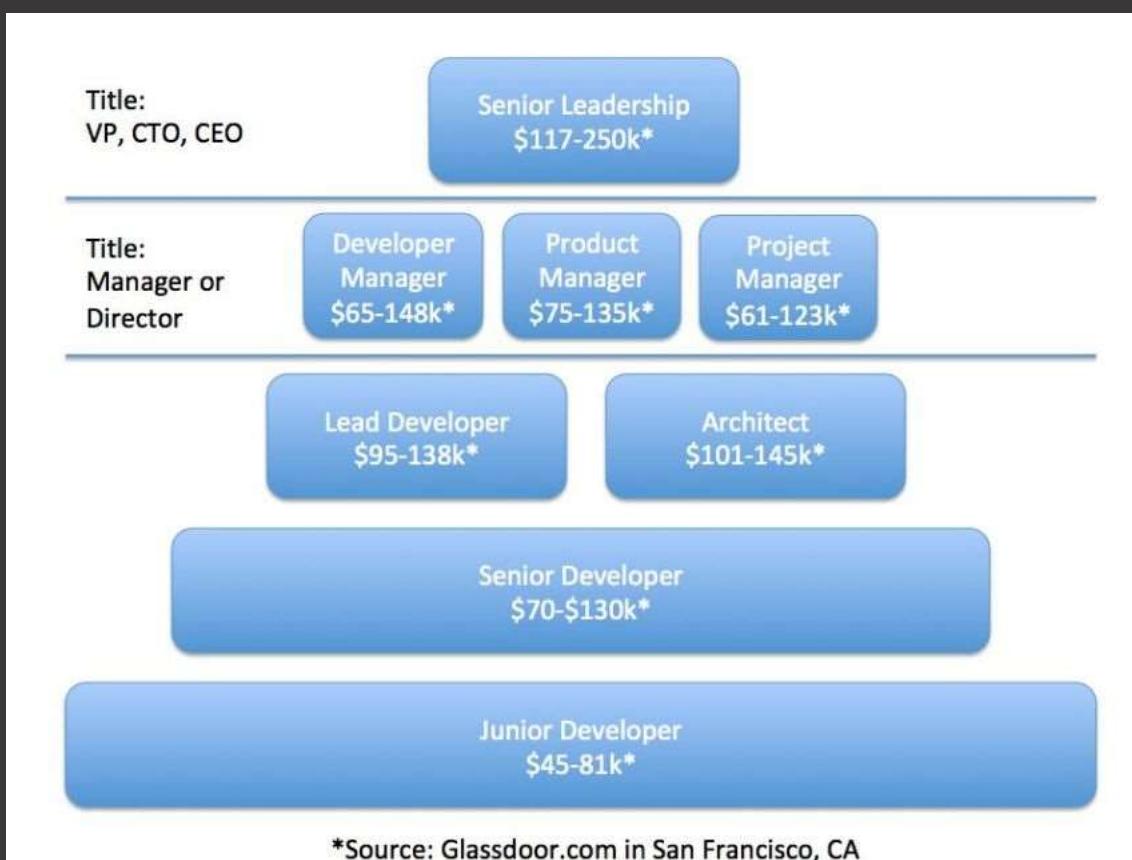
Như đã nói ở phần “tìm hiểu thị trường”, sau khi đã biết được những kỹ năng mà các nhà tuyển dụng yêu cầu, những ngôn ngữ hot hiện tại, các bạn cần bỏ thời gian để trang bị cho mình những kỹ năng đó. Hầu như các công ty nước ngoài đều có mức lương nhỉnh hơn công ty Việt Nam đôi chút, các bạn nên ráng đầu tư tiếng Anh.

Chúng ta cũng làm outsource khá nhiều cho các công ty phần mềm Nhật, bạn cũng có thể đầu tư cho tiếng Nhật thay vì tiếng Anh. Cố gắng kiểm cái bằng TOEIC, IELTS hoặc N3, N2, phụ cấp và lương đi kèm mấy bằng này cũng kha khá. (Nói không phải

khoe nhung mình TOEIC 940 mà IELTS 7.5, ko tin thì vào [Linkedin](#) của mình mà xem).

Xác định hướng đi

Nếu cảm thấy không có hứng thú hoặc không muốn gắn bó lâu dài với việc code, các bạn cũng có thể phát triển theo hướng BA (Business Analyst) hoặc QA. Con đường phát triển cho 1 developer cũng khá rộng mở. Nếu muốn tập trung vào code và technical, các bạn có thể đi theo hướng kĩ thuật: Senior Developer => Technical Lead => Software Architecture.... Nếu muốn làm việc với con người, muốn quản lý, các bạn có thể đi theo hướng quản lý: Senior Developer => Team Lead => Project Manager => Programming Manager ... Mỗi hướng đi đòi hỏi những kĩ năng riêng, mình sẽ giải thích rõ ở một bài viết khác.



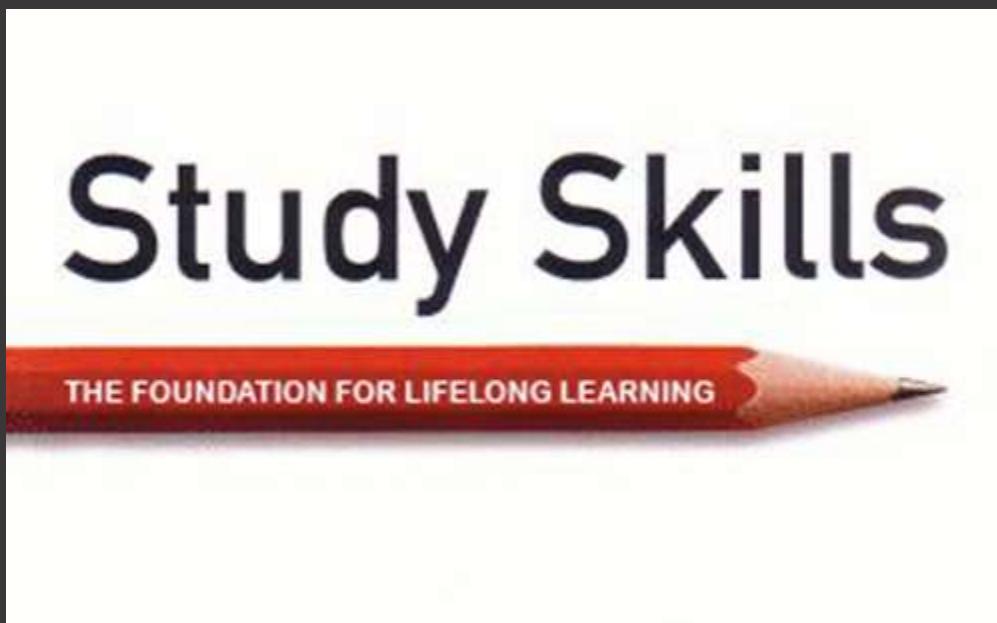
Phương pháp tự học, tự tìm câu trả lời

Không như các ngành khác, kiến thức trong ngành IT rất nhanh hết hạn. Với ngành xây dựng, xây cầu cách đây 50 năm cũng chẳng khác gì xây cầu bây giờ. Với ngành bác sĩ, bệnh cảm cách đây 50 năm triệu chứng cũng giống bệnh cảm bây giờ. Với

ngành IT, công nghệ, ngôn ngữ hoặc framework nổi tiếng cách năm 10-15 năm giờ chẳng ai xài nữa cả (Lúc đó còn mấy bác Nhật xài COBOL và VB).

Đó là lý do developer chúng ta phải học rất nhiều, học đủ thứ, học không ngừng nghỉ, để không lạc hậu với giới trẻ. Mình từng gặp 2 trường hợp 2 bác senior fail phỏng vấn vào cty mình, dù họ có 5-6 năm kinh nghiệm làm việc. Lý do là họ chỉ rành WinForm, WPF và WebForm, ... những công nghệ đang ngắt ngư, và có vẻ họ cũng không muốn học thêm công nghệ mới để tự làm mới mình. Trong truyện Kim Dung, Hồng Thất Công – bang chủ Cái Bang từng nói rằng : Trong võ học, không tiến túc là lùi. Mình nghi ngờ ông từng tốt nghiệp khoa CNTT của trường nào đó, vì câu nói này cũng khá đúng với ngành IT.

Báo chí vẫn phát ngôn nhanh nhảm rằng: Sinh viên Việt Nam thiếu khả năng tự học. Mình không muốn biện minh gì thêm, vì mình vẫn thấy nhiều bạn nhờ giải bài tập hộ, làm đồ án hộ trên các group facebook. Thiết nghĩ các trường nên hướng dẫn các sinh viên các tự học, cách nghiên cứu, cũng như rèn tính tự lập tự giác cho sinh viên. Mình cũng sẽ dành 1 bài viết riêng để hướng dẫn các bạn cách tiếp cận – tự học 1 ngôn ngữ hay công nghệ mới, bản thân mình từng áp dụng và thấy khá có ích.



Bài viết đã khá dài, xin kết thúc tại đây cho các bạn dễ đọc. Ở [phần cuối](#), mình sẽ chia sẻ một số kinh nghiệm về cách phát triển sự nghiệp, đạt được thành công trong công việc (1 phần là tự rút ra, một phần là đọc sách và học được trên pluralsight) mong các bạn đón xem.

Những điều trường đại học không dạy bạn – Phần 3

Posted on 11/06/2015 by Phạm Huy Hoàng

Đây là phần cuối trong series bài viết 3 phần “Những điều trường đại học không dạy bạn”:

1. Kỹ thuật lập trình
2. Cách nâng cao giá trị bản thân
3. **Thành công và thăng tiến trong môi trường làm việc**

Trong phần cuối của loạt bài, mình sẽ nói về những điều mà bạn-nào-cũng-muốn-biết-nhưng-trường-học-không-dạy, đó là : **Cách thành công và thăng tiến trong sự nghiệp.** Mình tổng hợp những điều này một phần từ sách vở, một phần từ pluralsight, một phần nhờ sự được các anh senior, PM, team leader chia sẻ. Có thể chúng không đúng 100%, nhưng biết những điều này sẽ giúp con đường nghề nghiệp của bạn bằng phẳng và “dễ thở” hơn rất nhiều.

Làm sao để phỏng vấn và xin việc

Sinh viên vừa ra trường, muốn có tiền nuôi sống bản thân thì phải đi làm. Muốn đi làm thì phải có công việc, mà muốn có công việc thì phải đi nộp đơn xin việc. Thế nhưng nhà trường không bao giờ dạy bạn cách viết 1 CV cho đàng hoàng, cách chuẩn bị, trả lời phỏng vấn ra sao. Có vô số những điều thuở mình muốn biết từ thời là sinh viên mà không biết hỏi ai:

- Viết, trình bài CV như thế nào để thu hút nhà tuyển dụng
- Chuẩn bị gì trước khi đi phỏng vấn
- Quy trình phỏng vấn của các công ty IT
- Những câu hỏi thường gặp khi đi PV (OOP, Database, Web, ...)
- Cách thỏa thuận lương (Cái này khá quan trọng, nhiều bạn sinh viên giỏi nhưng mới ra trường, khi PV xong, lúc thỏa thuận lương thường bị “ép giá”. Người thỏa thuận lương với bạn thường là manager, hoặc HR, họ làm chuyện đó thường ngày rồi, nên việc bạn không thỏa thuận lại họ cũng không có gì lạ).
-

Trong phạm vi bài viết này, mình sẽ không nói kĩ về CV, những điều cần chuẩn bị khi PV..., bạn nào có hứng thú có thể đón đọc ở những bài viết sau nhé.



Văn hóa ứng xử nơi công sở

Ngành IT là một ngành khá dễ chịu. Bạn không lo mình gặp phải cảnh “ma cũ bắt nạt ma mới”, phải “bung trà rót nước” cho các cụ lão làng, hay lo chia phe phái, tụm năm tụm ba nói xấu lãnh đạo. Dân developer chúng ta tập trung nhiều vào kĩ thuật, do đó có kĩ thuật giỏi thì sẽ được tôn trọng. Tuy nhiên, cũng có 1 số điều không-được-học-ở-trường mà ta nên lưu ý:

- Đặc điểm của ngành phần mềm là làm việc theo nhóm, do đó các bạn đừng thu mình lại, chịu khó hòa đồng với team: Đá banh chung, chơi game chung, đi nhậu chung, mát xa chung Một khi đã thân thì làm việc chung sẽ dễ hơn rất nhiều.

- Muốn mọi người biết mình giỏi, hãy thể hiện và chứng tỏ mình giỏi. Đây là một môn nghệ thuật khá khó, khoe nhiều các bạn sẽ mang tiếng là “nô”, là “tỏ thái độ”, khoe ít người ta sẽ tưởng các bạn “không biết gì”.
- Thông cảm cho các bậc cấp trên. Dân lập trình chúng ta thường tưởng mình giỏi, và nghĩ rằng các cấp trên ngu chết mẹ, có biết gì về code hay lãnh đạo đâu. Thật ra câu này cũng không sai, ngày xưa manager cũng từng là coder như bạn, vì code giỏi bị đẩy lên vị trí manager, có ai dạy họ kỹ năng lãnh đạo đâu. Bị cuốn vào công việc quản lý, họ không có thời gian trau dồi kỹ năng code nữa. Với suy nghĩ “thông cảm” thay vì “coi thường”, bạn sẽ dễ chia sẻ tầm nhìn với manager hơn, cơ hội thăng tiến cũng cao hơn.
- Đừng coi thường những người code dở hoặc không biết code. Đôi khi ta thấy một người code không giỏi nhưng lại nhanh thăng tiến, rồi chửi thầm “Tại sao nó code như hạch mà lên chức nhanh thế, chắc do giỏi nịnh sếp”. Có khi là do họ rành văn hóa công sở + soft skill hơn bạn đấy.



Những phương pháp thương thảo

Trong suốt quãng đời làm việc, bạn sẽ phải thương thảo vô số lần: Thương thảo với team và team lead khi phân chia task, thương thảo với HR khi thỏa thuận lương, thương thảo với manager khi muốn chuyển team, muốn được tăng lương... Do đó việc trau dồi kỹ năng thương thảo sẽ giúp bạn tiến xa hơn trên con đường nghề nghiệp. Mình từng đọc 2 cuốn sách khá hay về thương thảo là: “**Đắc nhân tâm**” và “**Một đời thương thuyết**”, các bạn có thể tìm đọc.

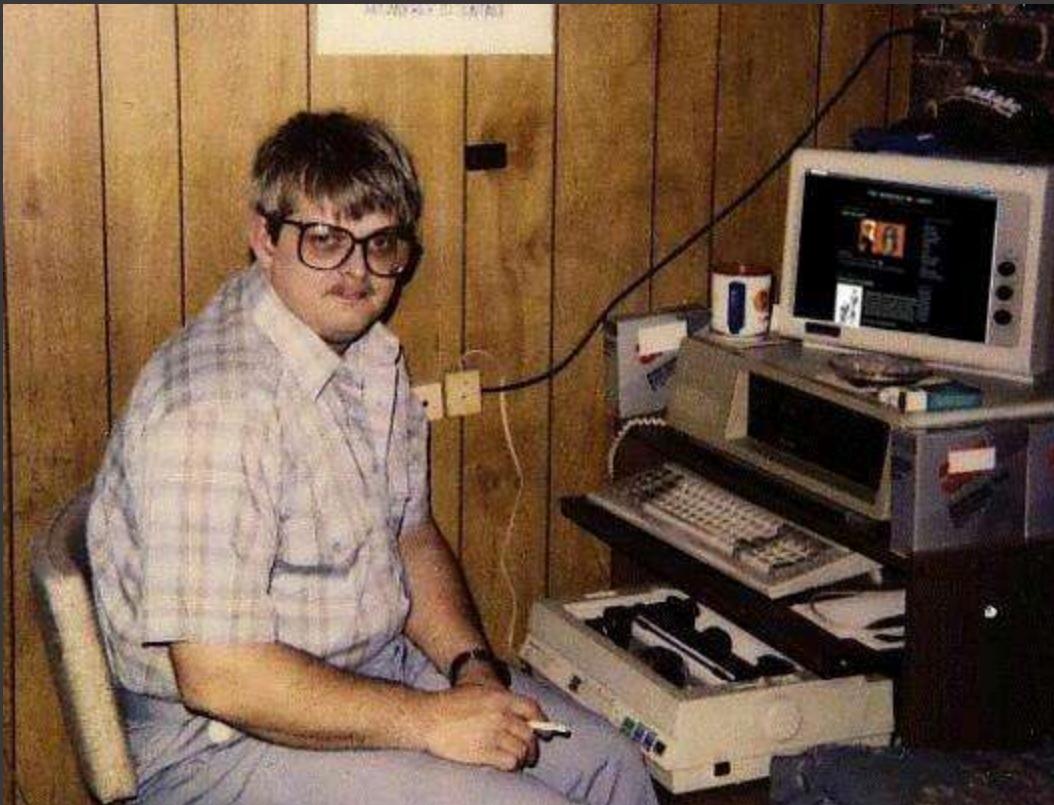


Cách xây dựng tiếng tăm (reputation)

Câu hỏi luôn đau đớn trong đầu mỗi người đi làm, đó là: Làm sao được tăng lương, làm sao được thăng chức, làm sao được leo lên vị trí cao hơn. Để làm được điều đó, bạn cần xây dựng tiếng tăm trong mắt đồng nghiệp, cấp trên. Làm như thế nào ư?

Một điều khá may mắn trong ngành IT là: Vì đây là một ngành nặng về kỹ thuật. do đó bạn chỉ cần giỏi tập trung trau dồi technical cho giỏi. Chỉ cần technical giỏi, bạn sẽ được đồng nghiệp coi trọng, cấp trên tin tưởng giao phó trách nhiệm. Chỉ cần technical giỏi, con đường sự nghiệp của bạn sẽ rộng thênh thang, bạn sẽ nhanh chóng leo lên vị trí senior, team leader, technical lead, ... Chỉ cần technical giỏi, lương bạn sẽ tăng vù vù, từ 500\$, 1000\$, 2000\$, các quảng cáo tuyển dụng toàn cầu người giỏi technical còn gì?

Các bạn vừa đọc vừa gật gù đồng ý với những điều mình viết trong đoạn trên? THẬT Á? TỈNH LẠI ĐI BẠN TRẺ À! **Con đường nghề nghiệp của 1 lập trình viên không đơn giản bằng phẳng như vậy đâu.** Chúng ta thường làm tưởng rằng: technical giỏi = lập trình viên giỏi. SAI BÉT.



Một lập trình viên cần nhiều hơn thế: kĩ năng lập trình, kĩ năng giao tiếp, kĩ năng thuyết trình, kĩ năng giải thích, ... Để lấy requirement từ khách hàng, bạn phải biết cách giao tiếp, biết cách hỏi, biết cách giải thích. Để thuyết phục mọi người trong nhóm làm theo mình, bạn cần rành kĩ năng chém gió, kĩ năng thuyết trình. Đó là lý do một số người technical không cứng nhưng vẫn lên được vị trí team leader, PM, nhờ họ giỏi “chém gió”, giỏi thuyết phục và quản lý người khác. Thậm chí, ở một số vị trí **thiên về technical** như: senior dev, technical lead,... bạn sẽ phải hướng dẫn developer mới, đưa ra solution và giải thích, những kĩ năng mềm này càng không thể thiếu được. Dưới đây là một số cách để xây dựng tiếng tăm. Một số cách mình được truyền lại, một số cách mình đã áp dụng, kết quả cũng khá là ok :

- Giúp đỡ đồng đội của bạn. Đừng lao vào ngoặt ý giúp đỡ, họ sẽ nghĩ là bạn ra vẻ, nhiều chuyện. Hãy thể hiện mình biết nhiều, khi có khó khăn họ sẽ thử nhờ bạn giúp. Dần dần khả năng technical + thái độ làm việc của bạn sẽ làm bạn được các thành viên khác nể trọng. Họ sẽ nhận xét tốt về bạn. và cấp trên đương nhiên sẽ để ý bạn hơn. (Lời khuyên: Hãy thành thật giúp đỡ người khác, đừng nghĩ mục đích của mình là vì tiếng tăm, bạn sẽ có được đồng đội lẫn bạn bè. Quá tập trung vào tiếng tăm sẽ phản tác dụng).

- Thể hiện với leader. Hãy estimate công việc nhiều một tí, sau đó hoàn thành sớm thời hạn. Sau khi hoàn thành công việc, hãy tự tìm việc làm hoặc báo cáo leader để giao việc. Leader sẽ đánh giá cao sự nhiệt tình + thái độ làm việc của bạn.
- Giữ thái độ khiêm tốn. Người khiêm tốn thường được tôn trọng hơn. Rất nhiều bạn SV mới ra trường thường giữ thái độ “ta đây giỏi”, làm gì cũng hất mặt lên trời, luôn bị đánh giá là “attitude không tốt”.
- Thực hiện seminar, thuyết trình, giới thiệu công nghệ. Khi bạn chia sẻ kiến thức của mình cho người khác, bạn sẽ nhận lại được kiến thức và sự nể trọng. Đó cũng là lý do mà mình viết [blog](#), lập tài khoản trên [stackoverflow](#),

Kết luận: Mình không có ý chê trách các trường đại học khi viết series này. Về bản chất, các trường chỉ dạy cho bạn kiến thức cơ bản, nó sẽ làm nền tảng giúp bạn học những điều mới dễ dàng hơn. Đó cũng là lý do mà mình viết blog này, để chia sẻ cho những bạn sinh viên còn đang học, mới ra trường hoặc vừa đi làm những điều mà **chỉ-đi-làm-rồi-mới-nhận-ra**. Muốn biết thêm về những điều này, hãy thường xuyên ghé thăm và đón đọc blog của mình nhé.

70 điều các developer giỏi thuộc nằm lòng – Phần 1

Posted on 28/05/2015 by Phạm Huy Hoàng

Ở bài trước, trong số sách đã giới thiệu, mình giới thiệu 1 cuốn sách gọi **The Pragmatic Programmer**. Như mình đã quảng cáo, cuốn sách này tập hợp rất nhiều kinh nghiệm được các tiền bối đúc kết lại qua bao nhiêu năm phát triển phần mềm. Vì sách khá dài, để tiện cho mọi người đọc, bác Jeff, chủ blog [codinghorror](#) đã rút gọn cuốn sách thành 70 điều dưới đây. 90% những điều này sẽ giúp ích cho sự nghiệp lập trình của bạn, do đó mình mạn phép dịch ra để chia sẻ lại. Bạn nào muốn có thể xem link gốc của cuối blog.

Danh sách 70 bài học đúc kết được từ **The Pragmatic Programmer**:

1. **Code có lương tâm:** Nếu đã quyết định lựa chọn sự nghiệp developer, hãy có bằng hết sức, đừng code dối hoặc làm xong cho qua việc.
2. **Code có suy nghĩ:** Hãy sử dụng đầu óc khi code. Biết tự đánh giá code của mình, khen khi code hay, chê khi code xấu.
3. **Đưa ra lựa chọn, dùng kiểm có:** Dùng kiểm có theo kiểu: “cái này không làm được”, hãy đưa ra lựa chọn kiểu : “Có thể ko làm được A, thay vào đó ta có thể làm B,C”.
4. **Đừng bỏ qua những lỗi vụn vặt:** Khi gặp các design tệ, code bựa,... hãy sửa ngay. Để lâu ngày, những lỗi nhỏ trong code sẽ tăng lên theo cấp số nhân.
5. **Sẵn sàng thay đổi:** Nếu không thể bắt mọi người thay đổi, hãy chỉ cho họ thấy những điều cần làm trong tương lai, cũng như nhờ họ góp sức thực hiện.
6. **Giữ cái nhìn toàn cục:** Dùng quá đẽ tâm tới chi tiết, hãy giữ cái nhìn tổng thể.
7. **Chất lượng cũng là requirement:** Hãy để chính người dùng góp sức vào việc đánh giá chất lượng của phần mềm.
8. **Không ngừng học hỏi:** Hãy biến việc học (ngôn ngữ mới, kiến trúc mới ...) thành thói quen.
9. **Đừng tin những gì mình nghe và đọc:** Dùng chạy theo số đông, chạy theo công nghệ. Hãy lựa chọn, phân tích dựa trên kiến thức bản thân + cấu trúc dự án.

10. **Điều bạn nói và cách bạn nói đều quan trọng như nhau:** Không cần biết bạn điều bạn nói hay tới mức nào, nếu bạn không biết cách diễn đạt nó ra, tất cả chỉ là vô nghĩa.
11. **Đừng lặp lại:** Với code cũng như document, hãy tập trung chúng lại **1 và chỉ 1** chõ.
12. **Tái sử dụng:** Tìm mọi cách để thứ bạn làm ra dễ dàng tái sử dụng (tạo library, v...v), mọi người sẽ tái sử dụng nó.
13. **Component nên tách biệt:** Mỗi component nên được thiết kế chỉ nhằm 1 mục đích duy nhất, không nên phụ thuộc vào component khác.
14. **Mọi thứ luôn thay đổi:** Mọi thứ đều có thể thay đổi: Requirement, design, công nghệ sử dụng hãy luôn sẵn sàng cho những thay đổi đó.
15. **Thử và sai:** Đôi khi requirement, kiến trúc chưa rõ ràng, đừng ngại thử và sai. Nếu bạn có sai, ít ra nó cũng sẽ đưa bạn đến gần cái đúng.
16. **Dùng prototype để tìm hiểu:** Như trường hợp trên, đôi khi ta dựng 1 prototype để tìm hiểu công nghệ, hoặc tìm hiểu requirement. Thứ ta thu được là kiến thức ta đã tìm hiểu, chứ không phải là prototype đó.
17. **Lập trình theo từ ngữ chuyên ngành:** Khi lập trình một hệ thống, hãy làm cho code và design của bạn giống từ ngữ thuộc lĩnh vực của hệ thống đó. VD như khi code hệ thống cho 1 trường học, ta dùng các class teacher, student, cho rạp chiếu phim thì dùng class cinema, ticket.
18. **Tập ước lượng:** Hãy thử ước lượng đánh giá thời gian, công sức bỏ ra khi code. Quá trình ước lượng này sẽ giúp bạn phát hiện 1 số vấn đề có khả năng phát sinh.
19. **Ước lượng dựa theo code:** Sau 1 thời gian code, bạn sẽ có thể ước lượng thời gian code chính xác hơn, dựa theo các số liệu trong quá khứ.
20. **Lưu trữ lại kinh nghiệm và kiến thức:** Trong quá trình code, team và bạn sẽ học được 1 số kinh nghiệm, 1 số trò “chích choá” để code chạy. Hãy lưu những kinh nghiệm này dưới dạng văn bản, trong tương lai nó sẽ có ích khi tìm lỗi, hoặc hướng dẫn thành viên mới.
21. **Hãy học cách dùng command shell:** Hầu như mọi chương trình: Window, Visual Studio, Git, v...v đều có cửa sổ command. Các cửa sổ này khá mạnh

mẽ, có thể thực hiện những chức năng mà UI ko làm được thông qua các câu lệnh.

22. **Sử dụng Text Editor thành thạo:** Bạn có thể code bằng nhiều editor: Notepad++, VIM, VS,... không cần biết nó là editor nào, hãy học cách dùng nó thành thạo (Sử dụng hotkey, v...v).
23. **Luôn luôn sử dụng Source Code Control (Git, SVN):** Hiện tại 90% bạn nào cũng biết cái này rồi, ko cần nhắc lại.
24. **Gặp bug thì fix:** Không cần biết bug là lỗi của bạn hay của ai khác, nếu nó cần được fix thì hãy fix trước đã, truy cứu trách nhiệm tính sau.
25. **Dùng não khi debug:** Khi debug, nhất là những lúc bị deadline dí, ta thường hay rối. Hãy HÍT SÂU, THỎ NHE, suy đoán nguyên nhân gây bug, sau đó bắt đầu tìm.
26. **Nguyên nhân gây lỗi:** Khả năng rất thấp là lỗi nằm ở hệ điều hành hoặc compiler, cũng như cái library (Điều này hiện tại ko đúng lắm, vì open-source mọc lên nhu nấm, không test hết được). Do đó, khi fix lỗi, hãy tập trung vào code do team mình tự code trước.
27. **Đừng dự đoán, hãy chứng minh:** Bạn đoán rằng transaction sẽ chạy mất 3 giây, database phục vụ 1000 users ... Đừng dự đoán suông, hãy dùng code, chạy test, đo đạc, tính thời gian để chứng minh điều bạn đoán.
28. **Học một ngôn ngữ xử lý text:** Hàng ngày bạn làm việc với text, sau không viết code để máy tính xử lý chúng giúp bạn.
29. **Viết code để tạo code:** Tương tự ý trên, nếu thấy code mình viết lặp đi lặp lại, sau ko tạo 1 chương trình tự sinh code (Hiện tại các IDE như Visual Studio, Eclipse đều hỗ trợ việc tự sinh code này).
30. **Không có phần mềm hoàn hảo:** Không có phần mềm nào hoàn hảo. Tuy vậy, hãy giảm thiểu những lỗi mà code hoặc người dùng có thể gặp phải
31. **Design và làm theo design:** Hãy chắc chắn rằng code của bạn **chỉ làm những điều mà nó phải làm**, ko hơn không kém. (VD code gửi mail active thì đừng block account, code mua hàng thì đừng block thẻ của người dùng).
32. **Chương trình nên crash:** Một chương trình crash sẽ gây thiệt hại, tuy nhiên thiệt hại không nhiều bằng một chương trình chạy sai, chạy nhầm.

33. **Sử dụng Assert:** Dùng Assert để tránh những lỗi có thể xảy ra trong code (Assert là một kiểu giống như throw Exception, ngày xưa thường hay dùng. Gần đây thì mình chỉ thấy Assert trong unit test).
34. **Sử dụng Exception:** Chỉ dùng Exception đúng lúc, đúng chỗ, dùng lung tung sẽ khiến code thành spaghetti code.
35. **Hãy code “có đầu có cuối”:** Nhớ giải phóng resource sau khi bạn đã dùng xong (Mở connection thì phải đóng, lấy bộ nhớ thì phải clear). Hiện tại 1 số ngôn ngữ bậc cao đã tự làm việc này, tuy nhiên lâu lâu nhớ + cẩn thận vẫn hơn.
- 35 điều tiếp sau sẽ được đăng trong [phần 2](#) nhé, mong các bạn đón xem.

Bản gốc: <http://blog.codinghorror.com/a-pragmatic-quick-reference/>

70 điều các developer giỏi thuộc nằm lòng – Phần 2

Posted on 02/06/2015 by Phạm Huy Hoàng

Nối tiếp [phần 1](#), bài viết này bao gồm 35 bài học tiếp theo trong số 70 bài học rút ra từ cuốn [The Pragmatic Programmer](#) mà mình đã nhắc tới ở bài trước.

35 bài học còn lại:

1. **Giảm thiểu quan hệ giữa các module:** Tránh sự chồng chéo, dính chùm bằng cách hạn chế việc các module gọi nhau quá nhiều (Nên đưa ra interface v...v).
2. **Config chử đừng tích hợp:** Hiện thực theo cách mà ta có thể tích hợp các lựa chọn về công nghệ 1 cách dễ dàng (Bằng cách config), chứ ko phải dùng code hay tích hợp bằng tay.
3. **Tập trung vào abstraction:** Code cần tập trung giải quyết theo hướng tổng quát, trừu tượng (Mình thấy cái lời khuyên này hơi vô lý, khá khó thực hiện).
4. **Nghiên cứu kĩ luồng chạy, tăng tính phân luồng của code.**
5. **Thiết kế theo service:** Mỗi component nên nằm trong 1 service riêng, giao tiếp với nhau thông qua interface (Đây là ý tưởng mở đầu cho khái niệm Dependency Inversion trong [SOLID](#)).
6. **Thiết kế cho việc phân luồng:** Thiết kế để hệ thống có thể chạy được phân luồng nếu yêu cầu.
7. **Tách biệt giữa view và model:** Tách biệt view và model ra sẽ làm tăng tính linh hoạt của thiết kế lên rất nhiều.
8. **Dùng design pattern blackboards để quản lý các luồng chạy.**
9. **Đừng code kiểu “hên xui”:** Đừng code theo kiểu “hên xui”, không biết vì sao code chạy, hoặc dùng các trò “chích choát”. (Nói vậy thôi, các bạn xem source code của Windows vẫn 1 đống trò hack hoặc chích choát đấy thôi).
10. **Đánh giá thuật toán:** Trước khi code, hãy thử đánh giá thuật toán/code bạn sử dụng sẽ chạy mất bao lâu.
11. **Thử nghiệm lại đánh giá:** Để xác thực đánh giá của mình, đừng tin vào thuật toán hay báo cáo, hãy chạy code trong môi trường thực tế và đo đạc lại.
12. **Refactor sớm và thường xuyên:** Code cũng giống như làm vườn, phải thường xuyên tia tót, thay đổi thiết kế thì mới tốt được. (Mình thì không đồng tình với

lời khuyên này lầm, đôi khi trễ deadline, code chết mợ, thời gian đâu mà refactor).

13. **Thiết kế để test:** Trước khi code, hãy nghĩ mình sẽ test thế nào.
14. **Test phần mềm cho kĩ, nếu không người dùng sẽ làm điều đó:** Test kĩ càng và quyết liệt, đừng để người dùng nhặt bug cho bạn.
15. **Cẩn thận với tool sinh code:** Hãy hiểu đồng code mà tool sinh ra trước khi sử dụng chúng (Một số bạn bây giờ khoái dùng code tự sinh mà ko hiểu, đến lúc có lỗi hoặc cần sửa chữa lại chả biết làm thế nào).
16. **Đừng tổng hợp requirement, hãy đào sâu mà tìm:** Requirement không có sẵn để bạn đi vòng vòng mà “tổng hợp”. Phải chịu khó đào sâu, làm phiền, quấy rối, tra tấn, móc họng khách hàng để họ “ói” requirement ra cho bạn.
17. **Làm việc với người dùng, đặt mình vào vị trí người dùng, suy nghĩ như người dùng.**
18. **Tập trung vào abstraction:** Những thứ trừu tượng như interface v...v sẽ sống lâu hơn implementation. Kể cả khi ta có thay đổi công nghệ hoặc thay đổi cách hiện thực, abstraction vẫn giữ nguyên.
19. **Hãy có 1 cuốn sổ tổng hợp:** Tập trung những từ ngữ chuyên biệt, ý nghĩa của chúng vào sổ. Sau này mỗi khi team cần tra v...v thì rất dễ dàng.
20. **Tìm cách giải quyết vấn đề “bất khả thi”:** Khi gặp vấn đề “bất khả thi”, hãy xác định các giới hạn, sau đó tự hỏi “Có bắt buộc phải làm cách này ko? Có thật sự phải giải quyết vấn đề này không?”
21. **Bắt đầu ngay sau khi bạn đã sẵn sàng.**
22. **Có nhiều thứ làm thì dễ hơn giải thích:** Đừng quá sa đà vào giải thích, mô tả, viết document, hãy code ngay khi có đủ dữ liệu bạn cần.
23. **Đừng quá gó bó vào phương pháp:** Đừng ép mình sử dụng các phương pháp lập trình một cách cứng nhắc, hãy áp dụng linh hoạt dựa theo khả năng của team và bản thân.
24. **Tool mặc tiền chưa chắc đã tốt.**
25. **Hãy chia team như chia code:** Đừng tách team ra thành các bộ phận như designer, tester, developer. (Đây là một trong những ý tưởng mở đầu cho kỉ nguyên Agile).
26. **Hạn chế làm tay:** Những process như copy file, chạy bản build, lặp đi lặp lại, chúng ta không nên làm bằng tay. Hãy lập trình để script làm điều đó.

27. **Test, test nǔa, test mãi:** Chạy test mọi nơi mọi lúc, với mọi bản build (Test nên là test tự động).
28. **Code chưa tính là xong cho tới khi toàn bộ test đã pass.**
29. **Vọc bug để kiểm tra test:** Hãy copy source code của bạn ra một vài bản khác, sau đó cố tình tạo bug để xem test của bạn có bắt được những bug đó ko.
30. **Test trạng thái của chương trình, không chỉ test code:** Đừng chỉ kiểm xem test có cover hết các dòng code hãy ko, hãy để ý xem test có cover được các trạng thái của chương trình hay ko
31. **Tìm bug 1 lần:** Tester chỉ test và tìm bugs 1 lần đầu tiên và duy nhất. Từ những lần sau, hãy để automation test chạy và verify bug đó.
32. **Tiếng Anh cũng chỉ là ngôn ngữ:** Hãy viết document như viết code: Dễ hiểu, đừng lặp lại, có thể sinh bằng tool nếu cần.
33. **Viết document, tách riêng với code:** Đừng để mỗi lần sửa code lại phải sửa document.
34. **Hãy vượt qua kì vọng của người dùng:** Code để chương trình đáp ứng kì vọng của người dùng, sau đó hơi quá kì vọng 1 chút, người dùng sẽ rất thích điều này.
35. **Kí tên trên code:** Nếu code làm bạn tự hào, hãy kí tên (Bằng comment). Nếu code như shit, bạn cũng nên kí tên, điều này giúp tăng tinh thần trách nhiệm của bạn với code mình viết ra.

Bài viết đến đây là hết rồi, rất mong nhận được sự góp ý, nhận xét từ các bạn (Ai góp ý comment gì đi, tui buồn là tui bỏ không viết blog nữa đó).

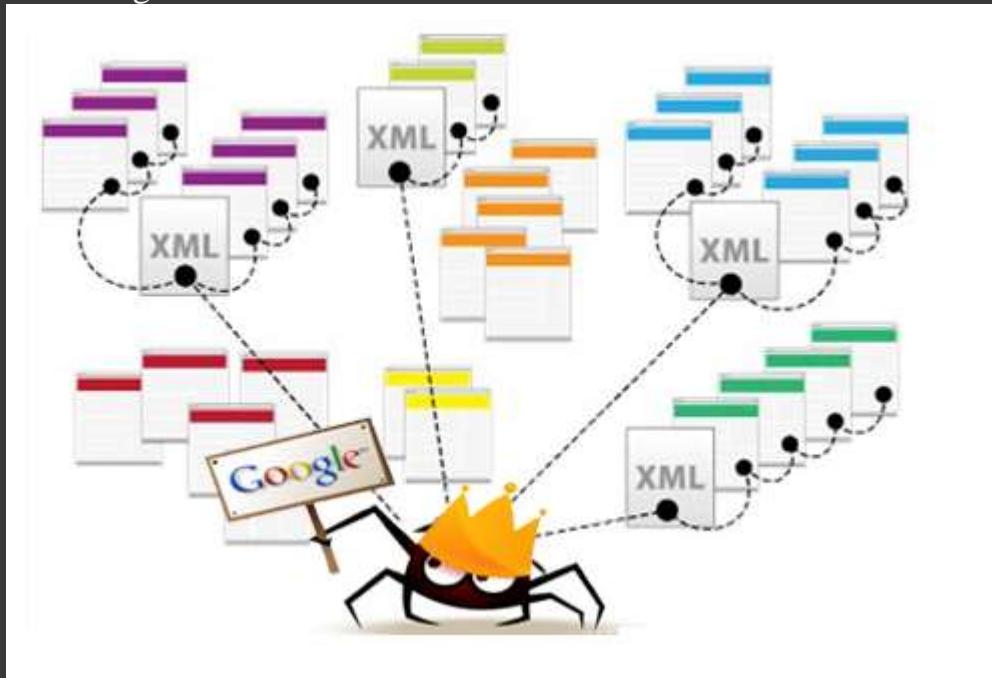
Bản gốc: <http://blog.codinghorror.com/a-pragmatic-quick-reference/>

[Tutorial] Trích xuất thông tin từ website với HTML Agility Pack

Posted on 28/07/2015 by Phạm Huy Hoàng

Đây là bài tutorial thứ 2 trên blog. Hiện nay, nhu cầu thu thập dữ liệu ngày càng tăng. Với một số trang như lớn như facebook, google, steam ta có thể sử dụng API do họ cung cấp để lấy dữ liệu. Trong nhiều trường hợp khác, ta thường trích xuất dữ liệu bằng tay (Mở trang web lên, copy dữ liệu vào file word, excel v...v), việc này vừa cực, vừa mất nhiều thời gian và công sức

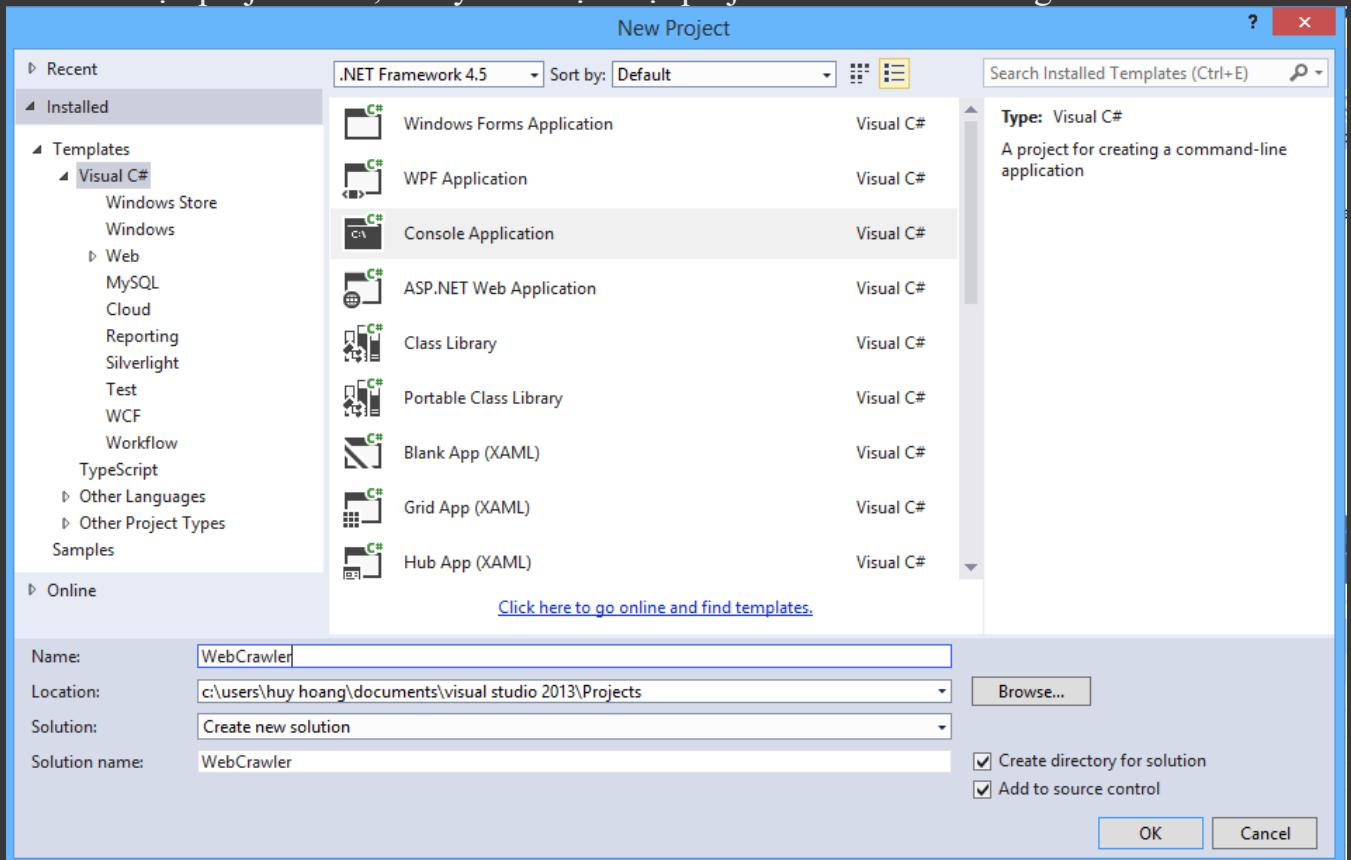
Đặt tình huống cụ thể, bạn muốn làm một ứng dụng đọc báo, lấy thông tin từ chuyên mục “[Đọc báo giúp ban](#)” trên [webtretho.com](#). Đây là một trang forum khá to, và dĩ nhiên là không có API để lấy dữ liệu. Ở đây, ta không thể lấy dữ liệu bằng tay được. Giải pháp duy nhất cho chuyện này là viết một phần mềm trích xuất dữ liệu từ bản thân trang webtretho.



Mình sẽ hướng dẫn các bạn trích xuất bằng thư viện [HTMLAgilityPack](#) và [Fizzler](#). HTMLAgilityPack là một thư viện parse HTML khá mạnh, lý do nó phổ biến là vì nó “chơi” được với hầu hết html, cả valid và unvalid (Trong thực tế thì số lượng website có HTML unvalid nhiều vô số kể, các thư viện khác sẽ dễ bị lỗi, HTMLAgilityPack

thì không). Kiến thức ở bài này **sẽ khá hữu dụng nếu sau này bạn cần trích xuất thông tin từ website** khác. Bạn có thể google thêm với từ khóa: web crawler. Cùng bắt tay vào làm nhé.

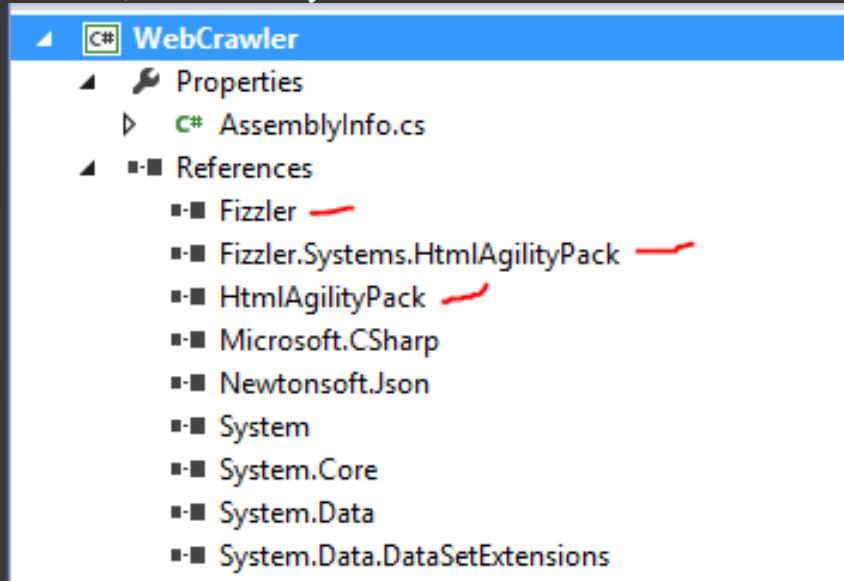
Bước 1: Tạo project mới , ở đây mình tạo một project console cho đơn giản.



Bước 2: Vào Tools -> Library Package Manager -> Package Manager Console. Đánh câu lệnh sau để cài đặt thư viện:

Install-Package Fizzler.Systems.HtmlAgilityPack

Sau khi cài đặt, nếu bạn thấy có đủ 3 reference như hình dưới là ok nhé.



Bước 3: Xem xét HTML của trang cần trích xuất. Ở đây, chúng ta sẽ trích xuất tên, link tới các topic trong diễn đàn, cũng như lấy số lượng view của topic đó.

Topic Title	Poster	Date	Lượt đọc	Trả lời
Hành phúc của người đồng tính Mỹ khi được công nhận hợp pháp	cau-vong-sau-mua	27/06/2015	189	0
Zalo 'mở cửa' cho những chiêu lừa đảo tinh - tiễn	xatare	27/06/2015	1.63K	0
'Ngày thử sáu kinh hoàng' của IS làm rùng động thế giới	xatare	27/06/2015	517	0
Thế giới 24h: Khủng bố dâm máu cả ba châu lục	xatare	27/06/2015	283	0
Trang Ha: "Chẳng có lòn chò, con lợn đâu mà phải giữ chân"	ngaymaiconhang	27/06/2015	7.22K	69
Mưa đá xuất hiện ở TPHCM	ME WINNI	27/06/2015	653	2

Sử dụng Developer Tool của chrome, ta sẽ thấy mỗi topic là 1 tag li, nằm trong 1 tag ul, có id là “threads.” Ta sẽ trích xuất dữ liệu từ những thẻ này. (**Kinh nghiệm của mình là bạn nên chọn tag dựa theo id, nằm gần dữ liệu mình cần lấy nhất.** Vì mỗi tag trong html chỉ có 1 id duy nhất, không bị trùng, ta dễ chọn tag và lọc hơn).

Trả lời mới nhất

Đọc nhiều nhất

Bài viết mới nhất

Trả lời nhiều nhất

Hạnh phúc của người đồng tính Mỹ khi được công nhận hợp pháp

cau-vong-sau-mua

Trả lời mới cau-vong-sau-mua 27/06/2015

189 0

Lượt đọc Trả lời

Zalo mờ cửa cho những chiêu lừa đảo tinh - tiễn

xatare

Trả lời mới xatare 27/06/2015

1.63K 0

Lượt đọc Trả lời

'Ngày thứ sáu kinh hoàng' của IS làm rúng động thế giới

vatare

Trả lời mới vatare 27/06/2015

517 0

Lượt đọc Trả lời

Events Network Sources Timeline Profiles Resources Audits Console

Styles Computed Event List

```
ul#threadlist li#thread_2073422.threadbit_nam_fix_select { font-weight: normal; width: auto; display: inline; font-size: 14px; color: #3474AB !important; }
```

```
#ctrlSub blockquote a, .infoAuNotice p a, .ctn_newRplSub i a, .welforum a { color: #186378; }
```

```
h1, h2, h3, h4, h5, h6 { Find in Styles }
```

Vào sâu hơn, ta sẽ thấy link và tiêu đề được lưu trữ trong tag a, có class là title, còn số lần đọc được lưu trữ trong tag b. Với những thông tin này, chúng ta đã có thể bắt đầu trích xuất dữ liệu

```
    <div class="con_cyprost">
        ▼<div class="titleTypPost">
            ▼<div style="float: left; width: 470px;">
                ▶<a class="title" href="http://www.webtretho.com/forum/f26/hanh-phuc-cua-nguo-dong-tinh-my-khi-duoc-cong-nhan-hop-phap-2073422/" id="thread_title_2073422" title>
                    <h3>Hạnh phúc của người đồng tính Mỹ khi được công nhận hợp pháp</h3>
                </a>
            </div>
            <span class="checkTypPost"> </span>
        </div>
        ▶<div class="auTypPost">...</div>
    </div>
    ▼<div class="folTypPost">
        ▼<ul>
            ▼<li style="width:58px">
                ▶<b>189 </b>
                <span>Lượt đọc</span>
            </li>
            ▶<li style="width:57px">...</li>
        </ul>
    </div>
```

Bước 4: Bắt đầu parse dữ liệu. Trước khi viết code, mình xin giới thiệu 1 số object, method của HTML AgilityPack mà các bạn nên biết:

–*HTMLDocument*: Đây là một class chứa thông tin về một file html (encoding, innerhtml). Ta có thể load dữ liệu vào HTMLDocument từ 1 URL hoặc từ 1 file. Trong bài này mình sẽ load từ url của webtretho.

–*HTMLNode*: Một HTMLNode tương đương với một tag (li, ul, div, ...) trong HTML. Node lớn nhất chứa toàn bộ tất cả sẽ là *DocumentNode*. Một số property của *HTMLNode* mà ta hay sử dụng:

- *Name*: Tên của node (div, ul, li).
- *Attributes*: Danh sách các attribute của note (Attribute là các thông tin của node như: src, href, id, class ...)
- *InnerHTML*, *OuterHTML*: Đọc tên là hiểu rồi nhỉ
- *SelectNodes(string xPath)*: Tìm các node con của node hiện hành, dựa trên xPath đưa vào.
- *SelectSingleNode(string xPath)*: Tìm node con đầu tiên của node hiện hành, dựa trên xPath đưa vào.
- *Descendants(string xPath)*: Trả ra danh sách các *HTMLNode* con của node hiện tại.

Đầu tiên, chúng ta sẽ sử dụng method *SelectNode*, sử dụng xPath để tìm node. Nếu bạn không rành, không nhớ hoặc không biết xPath thì đừng lo, phía dưới sẽ có cách khác dễ hơn.

```
1 HtmlWeb htmlWeb = new HtmlWeb()
2 {
3     AutoDetectEncoding = false,
4     OverrideEncoding = Encoding.UTF8 //Set UTF8 để hiển thị tiếng Việt
5 };
6 //Load trang web, nạp html vào document
7 HtmlDocument document = htmlWeb.Load("http://www.webtretho.com/forum/f26/");
8
9 //Load các tag li trong tag ul
10 var threadItems =
11     document.DocumentNode.SelectNodes("//ul[@id='threads']/li").ToList();
12
13 var items = new List<object>();
14 foreach (var item in threadItems)
15 {
16     //Extract các giá trị từ các tag con của tag li
17     var linkNode = item.SelectSingleNode("//a[contains(@class, 'title')]");
```

```

18     var readCount =
19     item.SelectSingleNode("./div[@class='folTypPost']/ul/li/b").InnerText;
20
21     items.Add(new { text, readCount, link });
22
23

```

Kết quả thu được khá là ưng ý:

```

foreach (var item in threadItems)
{
    var link = item.SelectSingleNode("./a[@class='title']").Attributes["href"].Value;
    var text = item.SelectSingleNode("./a[@class='title']").InnerText;
    var readCount = item.SelectSingleNode("./div[@class='folTypPost']/ul/li/b").InnerText;
    var obj = new { text, readCount, link };
    items.Add(obj);
}

```

Items Count = 50

- ▶ [0] { text = "Vụ "tu thế ngũ khô đ" và cái kết bất ngờ của các "anh hùng bàn phím"', readCount = "654 ", link = "http://www.webtretho.com/forum/f26/vu-tu-the-ngu-kho-do-va-cai-ket-bat-ngo-cl..." }
- ▶ [1] { text = "Điều gì đã xảy ra khi các quốc gia công nhận hôn nhân đồng giới", readCount = "6.99K ", link = "http://www.webtretho.com/forum/f26/dieu-gi-da-xay-ra-khi-cac-quoc-gia-cong-nhan-hon-nhan-do..." }
- ▶ [2] { text = "Tẩy chay có tổ chức: Nâng luồng hạt nhân của người tiêu dùng", readCount = "37.4K ", link = "http://www.webtretho.com/forum/f26/tay-chay-co-to-chuc-nang-luong-hat-nhan-cua-nguoi-tieu-dui..." }
- ▶ [3] { text = "Gần 1 triệu người kê khai tài sản, chỉ 4 người "không trung thực" (!)", readCount = "227 ", link = "http://www.webtretho.com/forum/f26/gan-1-trieu-u-nguo-i-ke-khai-tai-i-sa-n-chi-4-nguoi-i-khong-t..." }
- ▶ [4] { text = "Đám Vinh Hưng bắt ngô lên tiếng tố cáo Quang Lê "chơi bẩn"", readCount = "764 ", link = "http://www.webtretho.com/forum/f26/dam-vinh-hung-bat-ngo-len-tieng-to-cao-quang-le-cl..." }
- ▶ [5] { text = "Nhạc sĩ trẻ tung được đề cử giải "Cống hiến"; khẳng định M-TP không đạo nhạc", readCount = "1.1M ", link = "http://www.webtretho.com/forum/f26/nhac-si-tre-tung-duoc-de-cu-giai-c..." }
- ▶ [6] { text = "300 kg vải Việt bán lẻ hết sau 1 giờ tại Australia", readCount = "1.67K ", link = "http://www.webtretho.com/forum/f26/300-kg-vai-viet-ban-le-het-sau-1-gio-tai-australia-2074435/" }
- ▶ [7] { text = "Bắt ngô với nhan sắc 'tuột dốc không phanh' của dàn mỹ nhân Việt", readCount = "2.43K ", link = "http://www.webtretho.com/forum/f26/bat-ngo-voi-nhan-sac-tuot-doc-khong-phanh-cua-dan-my..." }
- ▶ [8] { text = "Bí dân trộm gach, 1/3 Vạn Lý Trường Thành "bốc hơi" hoàn toàn", readCount = "1.36K ", link = "http://www.webtretho.com/forum/f26/bi-dan-trom-gach-1-3-van-ly-truong-thanh-boc-ti..." }
- ▶ [9] { text = ""Game of Thrones" trở thành loạt phim ăn khách nhất trên HBO", readCount = "56.3K ", link = "http://www.webtretho.com/forum/f26/game-of-thrones-tro-thanh-loat-phim-an-khach-n..." }
- ▶ [10] { text = "Bắt ngô xuất hiện người tố ông Chấn giết người, Chung vồ tội", readCount = "987 ", link = "http://www.webtretho.com/forum/f26/bat-ngo-xuat-hien-nguoi-to-ong-chan-giet-nguoi-chung-vo-toi-2l..." }
- ▶ [11] { text = "Nước giếng chuyển màu tím đen khi pha trà", readCount = "102 ", link = "http://www.webtretho.com/forum/f26/nuoc-gieng-chuyen-mau-tim-den-khi-pha-tra-2074853/" }
- ▶ [12] { text = "Nữ sinh phi Nam đầu tiên đạt 9/9 IELTS", readCount = "1.24K ", link = "http://www.webtretho.com/forum/f26/nu-sinh-phi-a-nam-da-u-tien-da-t-9-9-ielts-2074358/" }
- ▶ [13] { text = "Hoa hậu Brazil mất vương miện vì đã có chồng", readCount = "112 ", link = "http://www.webtretho.com/forum/f26/hoa-hau-brazil-mat-vuong-mien-vi-da-co-chong-2075010/" }
- ▶ [14] { text = "Con nuôi Hoài Linh tham gia 'Giương mắt thần quen' mùa 2", readCount = "2.68M ", link = "http://www.webtretho.com/forum/f26/con-nuoi-hoai-linh-tham-gia-guong-mat-than-quen-mua-2-a-185..." }

Nhiều bạn sẽ cảm thấy cách này hơi khó. Thú thật là mình cũng không rành xPath cho lắm, vì vậy mình cũng không khoái cách này, do đó chúng ta có thể dùng LINQ to Object để tìm note. Code tương tự sẽ được viết như sau:

```

1 var threadItems = document.DocumentNode.Descendants("ul")
2             .First(node => node.Attributes.Contains("id") &&
3 node.Attributes["id"].Value == "threads")
4             .ChildNodes.Where(node => node.Name == "li").ToList();
5
6 foreach (var item in threadItems)
7 {
8     var linkNode = item.Descendants("a").First(node =>
9         node.Attributes.Contains("class") &&
10        node.Attributes["class"].Value.Contains("title"));
11     var link = linkNode.Attributes["href"].Value;
12     var text = linkNode.InnerText;
13     var readCount = item.Descendants("b").First().InnerText;
14
15     items.Add(new { text, readCount, link });
16 }

```

Chắc bạn hơi thật vọng phải không. Code bây giờ đã dễ hiểu hơn, không cần xPath xPiec gì. Tuy nhiên, code lại dài hơn, do ta phải dùng [lambda expression](#) và check null. Do một số node không có attribute class, **ta phải check null trước để tránh bị lỗi NullPointerException**. Còn cách nào hay hơn không nhỉ? Hãy kéo xuống dưới nhé, mình luôn để dành phần hay nhất ở dưới cùng.

Chú ý: Nếu bạn bỏ về check null ra sau, hàm chạy sẽ bị lỗi, để hiểu nguyên nhân hãy xem lại bài viết về [short-circuit](#) của mình nhé.

Bước 5: Cải tiến với Fizzler

Cả 2 cách trên đều làm bạn “đầu váng, mắt hoa”? Ok, mình cũng vậy :(. May mắn thay, còn một cách đơn giản hơn để select 1 node, đó là sử dụng Fizzler. Fizzler hỗ trợ CSS selector, cho phép ta sử dụng selector của CSS. Fizzler được mở rộng dựa trên HTMLAgilityPath, thêm 2 hàm sau vào HTMLNode:

+ *QuerySelectorAll*: Tìm các node con của node hiện hành, dựa trên css selector đưa vào.

+ *QuerySelector*: Tìm node con đầu tiên của node hiện hành, dựa trên css selector đưa vào.

Code bây giờ vô cùng đơn giản và dễ hiểu nhé

```
1 var threadItems = document.DocumentNode.QuerySelectorAll("ul#threads >
2 li").ToList();
3
4 foreach (var item in threadItems)
5 {
6     var linkNode = item.QuerySelector("a.title");
7     var link = linkNode.Attributes["href"].Value;
8     var text = linkNode.InnerText;
9     var readCount = item.QuerySelector("div.folTypPost > ul > li >
10 b").InnerText;
11     objs.Add(new { link, text, readCount });
12 }
```

Bước 6: Xuất kết quả ra đâu đó. Tới đây mọi chuyện đã xong, bạn có thể lưu kết quả vào file database hoặc xuất ra file text tùy mục đích sử dụng.

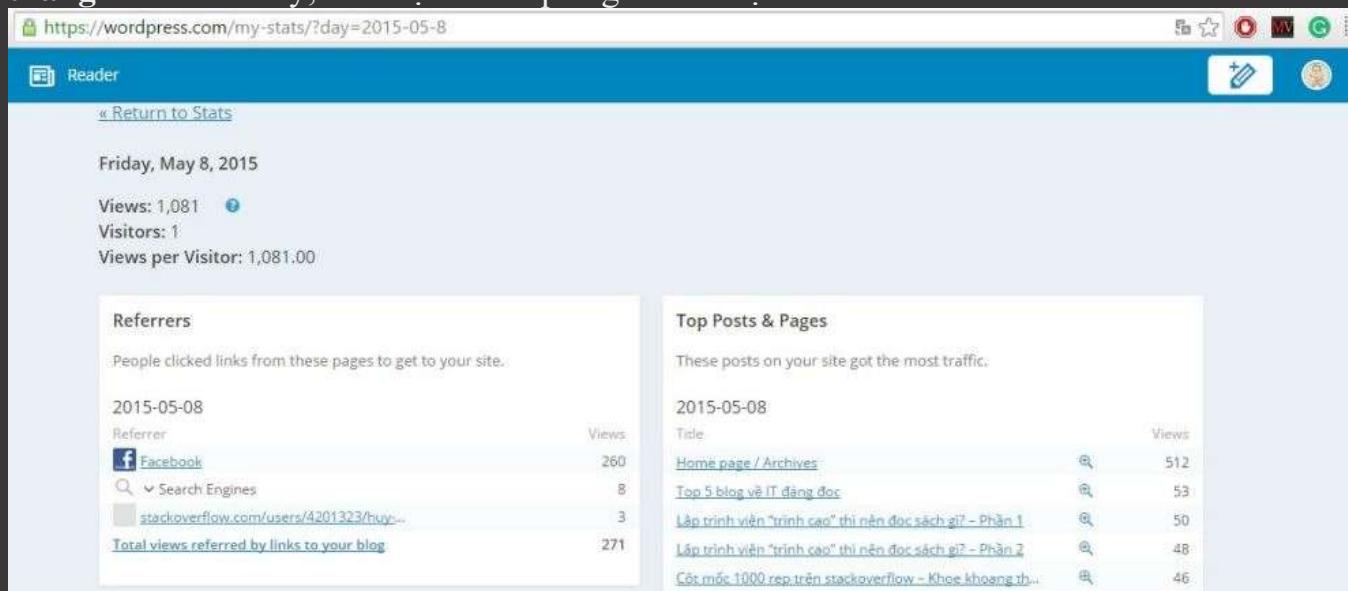
Bạn cũng có thể áp dụng thư viện này để trích xuất 1 số trang web bán hàng: hotdeal, muachung, ... Hi vọng bài viết này sẽ có ích cho sự nghiệp lập trình của các bạn.

Trải lòng với bài viết thứ 50 – Cảm ơn sự ủng hộ của mọi người

Posted on 23/07/2015 by Phạm Huy Hoàng

Mình bắt đầu viết blog này vào ngày 31/12 năm trước, thám thoát mà cũng đã được gần 8 tháng rồi nhỉ. Bài viết này ra đời nhân kỉ niệm lượng bài viết của blog đã đạt đến con số 50.

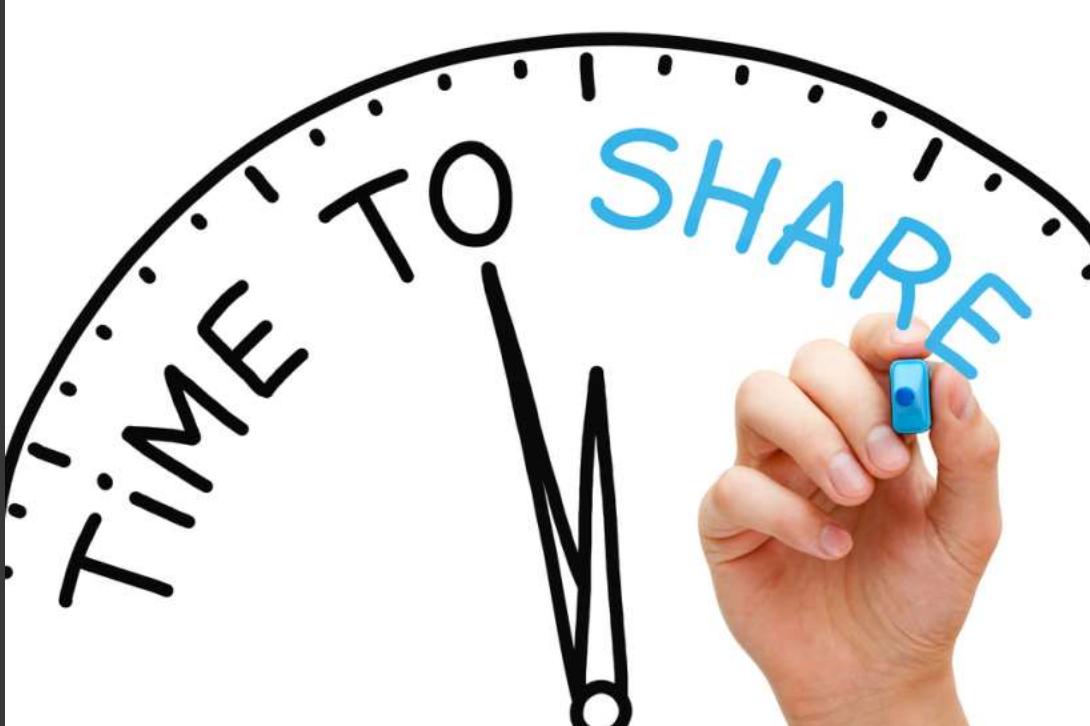
Bạn nào từng theo dõi blog chắc cũng thấy mình từng viết 1 bài viết ăn mừng **blog đạt được 1000 view** đầu tiên vào khoảng cách đây 2 tháng. Một chuyện dở khóc dở cười là, ngay sau khi mình chia sẻ và giới thiệu blog của mình với bạn bè trên facebook, lượng view trong ngày 09-05 đạt gần 1000 – bằng với **lượng view của blog trong 4 tháng**. Thé mới thấy, sức mạnh của quảng cáo bá đạo như thế nào.



Lúc mới lập blog, do lượng content chưa nhiều và thực sự hữu ích, mình chưa thực hiện PR và quảng cáo rầm rộ. Hiện nay lượng bài viết trên blog cũng đã kha khá, mình nghĩ nó sẽ giúp ích (không ít thì nhiều) cho các bạn. Nhờ quảng bá trên facebook, mình cũng có được một số bạn đọc trung thành, quen được nhiều bạn mới. Lượng view mỗi ngày tăng từ 20 cho đến 100-140, vô cùng cảm ơn sự quan tâm các bạn đã dành cho mình.

Như đã chia sẻ trong phần About me, mình viết blog này để chia sẻ kinh nghiệm lập

trình và phát triển nghề nghiệp cho các bạn sinh viên/ mới ra trường. Blog của mình vẫn sẽ giữ tôn chỉ như vậy trong tương lai. Nếu các bạn thấy một bài viết trên blog này có ích, hãy chia sẻ giới thiệu nó cho bạn bè nhé.



Mình tự cảm nhận thấy mình đã học được nhiều qua việc chia sẻ trên blog này: Mình không còn ngại khi phải viết những bài viết dài hơi, biết cách sắp xếp các ý khi diễn đạt, biết cách giải thích một vấn đề sao cho đơn giản dễ hiểu. Mình chân thành khuyên mỗi bạn lập trình hãy thử viết một blog nho nhỏ. Đừng nghĩ rằng: Mình có biết gì mà chia sẻ. Có khi những kiến thức, kinh nghiệm nho nhỏ mà bạn cho là vô dụng lại vô cùng quý giá với người khác đây.

Bạn không giúp được tất cả mọi người, nhưng đối với những người được giúp, họ sẽ thấy nó rất có ý nghĩa. (Bản thân mình rất vui vì được nhiều bạn khen bài viết hay, blog có ích). Minh được truyền cảm hứng từ câu chuyện nhỏ nhặt dưới đây của đương [Tony buổi sáng](#), nay chia sẻ lại cho các bạn.

Cứ mỗi chiều trên bãi biển nọ, người ta thấy một ông lão lang thang nhặt những con sao biển bị mắc cạn ném xuống biển. Ông làm việc này một cách cẩn thận. Nếu không, ngày mai khi nắng lên, chúng sẽ bị chết khô. Có người bảo việc làm của ông chỉ vô ích, vì mỗi ngày có hàng vạn con sao biển bị sóng đánh lên bờ, mà ông thì chỉ có hai tay hai chân và 24 giờ thôi, thì cả đời ông cũng không sao cứu hết chúng được.

Ông lão mới trả lời, cả cuộc đời tôi không thể nào cứu hết những con sao biển, nhưng đối với từng con sao biển được tôi cứu giúp, nó lại có một cuộc đời.



Dưới đây là tổng hợp 1 số series, bài viết hay, được nhiều người xem trên blog của mình. Chúng khá thú vị + hữu ích cho các bạn.

1. Series “[Học ngôn ngữ lập trình gì bây giờ](#)”: Cái nhìn tổng quát + lời khuyên của mình về ngôn ngữ lập trình mà các bạn sinh viên nên lựa chọn để gia tăng cơ hội nghề nghiệp.
2. Series “[Cách tiếp cận một ngôn ngữ/công nghệ mới](#)”: Hướng dẫn cách học một ngôn ngữ/công nghệ, thái độ cần có và những định hướng cho việc học tập của bạn.
3. Series “[Những điều trường đại học không dạy bạn](#)”: Những điều vô cùng quan trọng: kĩ thuật lập trình, nâng cao giá trị bản thân, xây dựng danh tiếng, ... mà bạn không-bao-giờ học được ở trường.
4. Series “[Muôn nẻo đường tìm việc](#)”: Kinh nghiệm viết CV và trả lời phỏng vấn.
5. Series “[Lật mặt na sự bá đạo của LINQ](#)”: Series này đề cập tới những yếu tố cấu thành LINQ: [Lambda Expression](#), [Generic](#), [Extension method](#), [Delegate](#), [IEnumerable](#). Đọc xong series này, kiến thức cơ bản của bạn sẽ vững hơn nhiều, đồng thời bạn sẽ có 1 cái nhìn tổng quan và rõ ràng về sự “màu nhiệm” của LINQ.

6. Series “[C# hay ho](#)“: Bao gồm những bài viết lặt vặt, tổng hợp những điều thú vị trong C#, có thể có nhiều người biết, hoặc biết không đầy đủ. Tuy nhỏ nhặt nhưng những kiến thức này khá hữu dụng.
7. [Con đường phát triển sự nghiệp cho developer](#).
8. [Luân về comment code \(Kiếm hiệp style\)](#)



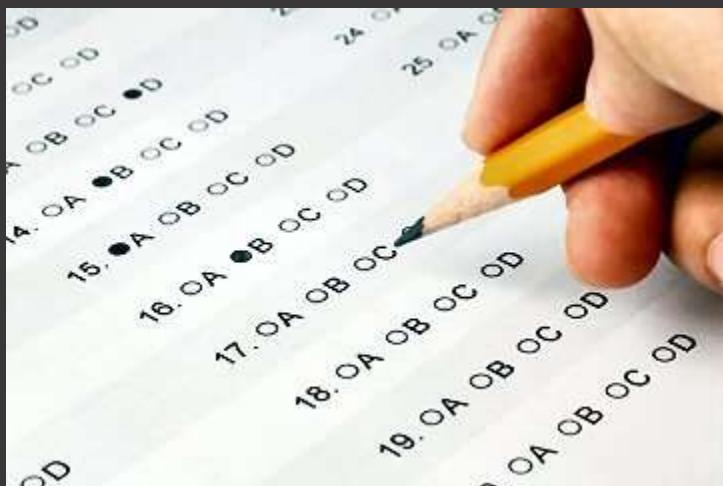
Ngoài ra, để ăn mừng post thứ 50 (cũng như tặng thưởng cho các bạn chịu khó đọc tới cuối bài), **2 bạn đầu tiên comment cho post** này sẽ có quyền yêu cầu mình viết một bài viết (về kĩ thuật, hoặc công việc) về lĩnh vực mà bạn muốn tìm hiểu nhé. Nếu có thắc mắc hay khó khăn gì, các bạn có thể gửi mail hoặc liên lạc trực tiếp qua facebook mình. Chỉ cần không phải là hỏi code hay nhờ fix bug, làm bài tập hộ thì mình rất sẵn lòng giúp đỡ.

Muôn nẻo đường tìm việc – Phần 2: Vượt qua kì phỏng vấn like a boss

Posted on 21/07/2015 by Phạm Huy Hoàng

Sau khi đọc xong [phần 1](#), hi vọng các bạn đã chuẩn bị được cho mình 1 mẫu CV rõ ràng mạch lạc. Nếu mọi chuyện đều ổn, khoảng 1-5 ngày sau khi gửi CV, bạn sẽ được một/nhiều công ty gọi điện thoại mời đi phỏng vấn. Sau khi nhận điện thoại, hãy kiểm tra hộp mail, sau đó **gửi mail xác nhận rằng mình sẽ có mặt tại công ty lúc X giờ, ngày Y để thực hiện phỏng vấn** nhé, quên gửi mail là chết đắng.

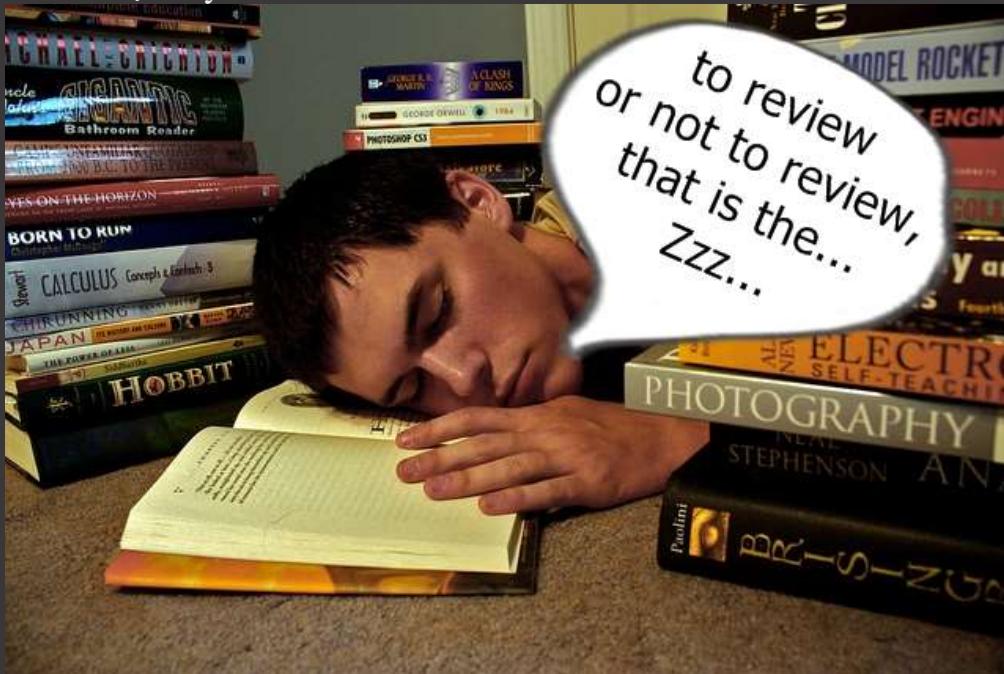
Một số công ty còn có thêm vòng interview qua điện thoại. Một số công ty lớn (Fsoft, Harvey Nash, ...) có cả entry test – bài thi đầu vào dành cho ứng viên, bao gồm: Thi tiếng Anh, kiến thức lập trình cơ bản, các bạn nên chú ý.



Những việc bạn cần làm trước khi đi PV

- **Tìm đường đến nơi PV:** Nếu có thể, hay đi ngang nơi phỏng vấn trước đó 1 ngày. Điều này giúp bạn ước lượng được khoảng thời gian đi. Tới hôm phỏng vấn, bạn sẽ khá hồi hộp, lo lắng; biết trước đường tới nơi phỏng vấn sẽ giúp bạn tự tin hơn so với tới hôm PV mới tìm.
- **Tìm hiểu công ty bạn PV:** Đây là điều **quan trọng nhất** mà khá nhiều bạn bỏ sót. Hãy lên trang web của cty đó, xem và ghi nhớ những thông tin quan trọng như: Công ty tập trung vào lĩnh vực nào, môi trường làm việc ra sao, công ty coi trọng những giá trị nào của nhân viên... Biết những điều này, bạn sẽ có cái nhìn tổng quan về công việc mình sắp làm, cũng như ghi điểm với nhà tuyển dụng.

- **Xem xét giá cả thị trường, yêu cầu lương hợp lý:** Có một thực tế phũ phàng là, **nếu bạn không đòi hỏi mức lương, nhà tuyển dụng sẽ trả cho bạn mức lương thấp nhất có thể** (Ai cũng muốn tiết kiệm mà). Mức lương có thể được ghi rõ trong mẫu quảng cáo việc làm, có thể không. Bạn có thể tham khảo mức lương từ những người quen làm công ty đó, hoặc người quen có vị trí tương đương vị trí bạn muốn apply (VD bạn muốn apply vị trí senior dev, bạn có thể hỏi người quen làm senior dev để biết khoảng lương).
- **Ôn lại kiến thức:** Xem lại kiến thức cơ bản về lập trình, OOP, tiếp theo là những kiến thức liên quan đến phần mà nhà tuyển dụng yêu cầu ([C#](#), Java, SQL, MVC, Struts, ...). Bạn có thể google *C# interview questions* hoặc tự đê tìm những câu hỏi phỏng vấn hay được hỏi. Bạn nên dành 3-5 ngày cho việc này.



Quy trình phỏng vấn

Như mình đã nói ở đầu bài, quy trình phỏng vấn ở các công ty thường khác nhau, tuy nhiên nó thường bao gồm các bước sau.

Phone interview hoặc entry test

Đây là vòng để sàng lọc ứng viên, tiết kiệm thời gian cho nhà tuyển dụng. Bạn sẽ bị hỏi một số kiến thức cơ bản về ngôn ngữ, framework mình ghi trong CV (C#,

MVC, ...). Qua buổi phỏng vấn ngắn (20-30p), người phỏng vấn sẽ quyết định có nên mời bạn đến cty phỏng vấn hay không

Face interview – Vòng quan trọng nhất

Nếu mọi chuyện êm xuôi, bạn sẽ tới công ty để phỏng vấn. Hãy nhớ rằng đây cũng là cơ hội để bạn phỏng vấn ngược lại nhà tuyển dụng. Bạn có thể quan sát công ty để có cái nhìn tổng quan về không khí làm việc, môi trường làm việc. Dưới đây là những câu hỏi bạn sẽ được hỏi khi phỏng vấn:

- *Giới thiệu bản thân?* Bạn có thể nói sơ về số năm kinh nghiệm, sở thích về công nghệ, vị trí muốn làm (Khoảng 2p thôi nhé).
- *Hãy nói về 1 project bạn đã làm? Bạn làm vai trò gì?* Người PV sẽ hỏi khá kỹ về cấu trúc project, công việc bạn làm, khó khăn bạn gặp phải, cũng như cách xử lý. Họ sẽ đánh giá được nhiều điều từ bạn qua câu hỏi này.
- *Một loạt các câu hỏi technical liên quan tới những gì bạn ghi trong CV.* Bạn sẽ được hỏi từ backend (Database, SQL, Entity Framework, LINQ, Delegate), cho tới front end (HTML, CSS, jQuery, AngularJS,...), và những điều liên quan tới framework (MVC routing, model mapping, ...). Ngoài những câu lý thuyết, có thể bạn sẽ được yêu cầu giải quyết vấn đề. Vd: Anh muốn submit một form bằng Ajax phải thế nào?
- *Một loạt câu hỏi liên quan đến OOP, kiểu dữ liệu, thuật toán v..v.* Đây là những câu hỏi về kiến thức fundamental, dùng để đánh giá kiến thức nền tảng của bạn, không liên quan đến ngôn ngữ hay framework nhé. Có thể bạn sẽ bị bắt viết code trên giấy hoặc trên máy để giải một số câu hỏi đánh giá trình độ.
- *Một số câu hỏi cá nhân để không khí bớt căng thẳng: Bạn có sở thích gì? Bạn có điểm yếu điểm mạnh gì?* Cứ trả lời thành thật nhé. Người PV không chỉ đánh giá bạn qua khả năng kỹ thuật, mà còn đánh giá qua thái độ làm việc, thái độ trả lời câu hỏi. Có nhiều câu hỏi bạn không biết, nhưng nếu cố gắng trả lời, thể hiện thái độ muôn học hỏi bạn vẫn sẽ được đánh giá cao nhé.
- Cuối buổi phỏng vấn, bạn sẽ được hỏi câu cuối cùng: *Bạn có câu hỏi gì không?* Mình khuyên các bạn nên hỏi về những điều sau đây: Môi trường làm việc ra sao, có bắt OT không? Chính sách review tăng lương tăng thưởng thế nào? Công ty có tổ chức seminar hay chính sách gì để giúp nhân viên phát triển

không?. Những câu hỏi này sẽ thể hiện bạn có tinh thần làm việc nghiêm túc, biết suy nghĩ đến tương lai.



Người PV bạn có thể là PM, Team Leader của một dự án. Nếu mọi chuyện ok, bạn sẽ phỏng vấn với quản lý, nhân sự hoặc chính người PV để **đàm phán về lương** bỗngcũng như thời gian bạn có thể bắt đầu đi làm. Sau khoảng 2-10 ngày làm việc, bạn sẽ nhận được điện thoại, mail thông báo kết quả PV. Nếu bạn được nhận, mail có thể đính kèm offer letter. Bạn nhớ chuẩn bị đầy đủ giấy tờ để nộp cho phòng nhân sự vào ngày đầu tiên đi làm của mình nhé.

* Sau buổi phỏng vấn, **nhớ gửi một email cảm ơn** cho người đã phỏng vấn mình. Đây là một điều nhỏ, hiệu quả lại lớn lớn mà các bạn thường “quên” không làm. Ngoài ra, mọi người có thể tìm đọc cuốn [What color is your parachute?](#). Đây là một cuốn sách khá hay về xác định bản thân, phỏng vấn và xin việc. Hi vọng bài viết này sẽ có ích cho các bạn.

Muôn nẻo đường tìm việc – Phần 1: Viết CV rõ ràng và chuyên nghiệp

Posted on 16/07/2015 by Phạm Huy Hoàng

Nhu đã nói trong bài viết: [Những điều trường đại học không dạy bạn](#), kĩ năng viết CV và phỏng vấn xin việc là những kĩ năng quan trọng không được dạy ở trường. Theo yêu cầu của một số bạn, mình viết bài này nhằm chia sẻ một số kiến thức về cách viết CV + xin việc cho các bạn sinh viên đang học hoặc mới ra trường.

Tìm việc

Ở Việt Nam, lập trình là một ngành khá dễ xin việc. Chỉ cần có khả năng code khá, các bạn đã có thể dễ dàng xin việc với mức lương tạm ổn so với các ngành khác, không cần phải quen biết, lót tay hay con ông cháu cha gì.

Các trang web dưới đây là những web site về việc làm lớn nhất Việt Nam hiện tại.

Đây là những kênh trung gian giữa các nhà tuyển dụng và người tìm việc, do đó mình khuyên các bạn đang tìm việc hãy tạo 1 CV online cho mỗi trang này:

- IT Việt: <https://itviec.com/>
- Vietnamworks: <http://www.vietnamworks.com/>
- Careerlink: <https://www.careerlink.vn/>
- Careerbuilder: <http://careerbuilder.vn/>
- Linkedin: <https://www.linkedin.com/>
- Một số group lập trình trên facebook cũng thường có các mẫu tin xin việc.

Lâu lâu vẫn có nhân sự công ty khác gọi mình mời phỏng vấn, không cần nộp gì do họ đã xem CV trên mạng rồi. (Do mình đang có việc làm nên đành từ chối, bạn nào muốn có thể liên hệ mình, mình sẽ giới thiệu cho họ, lợi cả 2 bên nhé). Cá nhân mình khuyến khích các bạn nên trau chuốt CV trên linkedin, chức năng connect của trang này cho phép bạn kết nối với nhiều người (đồng nghiệp, cấp trên), mở rộng mạng lưới nghề nghiệp của bạn.



Viết CV + nộp đơn xin việc

1. Nội dung

Do đã có vô số các bài viết hướng dẫn viết CV trên mạng rồi, mình sẽ không nhắc lại. Nếu bạn chưa biết viết CV như thế nào, hãy tìm một CV mẫu cho vị trí của mình (vd bạn là junior developer hãy xem CV mẫu của 1 junior dev, tương tự với junior QC), chỉnh sửa một tí cho phù hợp. Format CV của linkedin cũng khá ổn; chỉ cần bê các phần từ trên linkedin về, chỉnh sửa một chút là bạn đã có một CV khá chuẩn.

Một điều cần lưu ý: Hãy chỉnh sửa CV cho phù hợp với vị trí bạn ứng tuyển. Giả sử bạn biết cả C#, Java, Python, nhưng công việc bạn ứng tuyển đòi hỏi C# MVC, hãy để các kỹ năng và dự án liên quan tới C# lên đầu trong CV.

2. Cách trình bày

CV có thể không đẹp lộng lẫy, nhưng phải rõ ràng và ngắn gọn. Cỡ chữ nên phù hợp là 12-14, sử dụng 1 số font cơ bản như Times New Roman, Arial. Các phần đề mục nên viết rõ ràng, in đậm, nhìn lướt qua có thể đọc được các đề mục này. Dân IT chúng ta cũng không quá bất bé về mặt hình thức, nhưng các bạn nên chịu khó canh thẳng hàng thẳng lối, đồng thời soát kỹ các lỗi chính tả. Những điều nhỏ nhặt này sẽ thể hiện tính chuyên nghiệp và sự cẩn thận của bạn.

Các bạn cũng nên export file CV ra dưới dạng PDF, máy nào cũng mở được. Nếu để dạng Word có thể sẽ bể font, bể format, ảnh hưởng chất lượng CV. CV nên đặt tên theo format “[Tên vị trí] Tên bạn“, giúp nhà tuyển dụng dễ đọc và phân loại (Hoặc

bạn đặt theo format mà nhà tuyển dụng đưa ra trong quảng cáo tuyển người). Đây là CV của mình, không đẹp cũng không pro nhưng khá chỉnh chu. Các bạn có thể tham khảo và góp ý: [\[Developer\]\[Pham Huy Hoang\] Resume.pdf](#)



Một số sai lầm mà các bạn (và mình ngày xưa) từng gặp phải khi viết CV:

- Áo tưởng sức mạnh:** Không biết vô tình hay cố ý mà trong phần kĩ năng, tự đánh giá trình độ của mình là Intermediate, Expert, hoặc Master, dù mới ra trường, chỉ mới làm 1,2 cái đồ án, chưa làm dự án doanh nghiệp nào. Nếu đã từng làm qua 1 công nghệ, các bạn nên ghi thời gian đã tiếp xúc với công nghệ đó, vậy là đủ. Trình độ bạn ở mức nào thì người phỏng vấn sẽ tự xác định, bạn có nói mình là Master họ cũng không tin đâu.
- Gây chú ý không đúng cách:** Dùng font màu mè, font lạ để đập vào mắt nhà tuyển dụng. Cách này thường gây tác dụng ngược. Họ có thể quăng CV của bạn vào sọt rác không thương tiếc.
- Chém gió:** Mình từng gặp trường hợp có bạn chém gió về số năm đi làm + công nghệ mình biết trong CV. Tới lúc PV, bị văn hỏi thì bạn bảo là: mình làm

part time, công nghệ ABC gì đó mình hiểu nhưng chưa làm bao giờ ... Nhiều đó là quá đủ để rót dài rồi nhỉ. Hãy nhớ rằng mục đích của CV chỉ là **giúp bạn qua được vòng gửi xe, được mời phỏng vấn** thôi, chứ không giúp bạn có được việc làm ngay đâu nhé.



Ở [phần 2](#), mình sẽ chia sẻ một số **qui trình tuyển dụng** của các công ty, **các câu hỏi thường gặp**, cũng như **những điều cần lưu ý** khi đi phỏng vấn, mong các bạn đón đọc.

Một số câu phỏng vấn thú vị về lập trình

Posted on 14/07/2015 by Phạm Huy Hoàng

Mấy hôm gần đây, tình cờ mình đọc được cuốn sách: [Cracking the Coding Interview: 150 Programming Questions and Solutions](#). Đây là một cuốn sách khá hay; sách viết về những câu hỏi thường gặp trong các cuộc phỏng vấn, qui trình phỏng vấn của Yahoo, Google, Amazon, Facebook. Trong cuốn sách có rất nhiều câu hỏi về lập trình hay, mình chỉ chia sẻ một số câu mình cảm thấy đơn giản và thú vị. Bạn nào muốn tìm hiểu thêm hãy tự tìm ebook nhé.



Linked List (Chuỗi liên kết)

1. Hiện thực thuật giải tìm phần tử thứ n **tính từ cuối lên** của 1 linked list. (Đây là linked list chứ không phải double linked list nhé).

Chuỗi: a->b->c->d->e->f. n = 3, cho ra phần tử d.

2. Hiện thực thuật giải xóa một phần tử khỏi linked list, bạn chỉ được access vào phần tử đó?

Chuỗi: a->b->c->d->e->f. Đưa vào phần tử c, chuỗi còn lại a->b->d->e->f.

Dịch bit

3. Hãy cho biết đoạn code sau làm gì: $((n \& (n-1)) == 0)$.

Đố mèo

4. Sử dụng + - * / (), hãy làm biểu thức sau đúng: $3 \ 1 \ 3 \ 6 = 8$

5. Một bàn cờ vua 8×8 , bị cắt mất 1 ô góc trên bên trái, 1 ô góc dưới bên phải. Mỗi quân domino có thể che được 2 ô trên bàn cờ. Có thể dùng 31 quân domino để che hết 62 ô còn lại của bàn cờ không? Vì sao/Vì sao không?

Testing

6. Hãy tìm lỗi trong đoạn code sau

```
1  unsigned int i;
2  for (i = 100; i <= 0; --i) printf("%d\n", i);
```

Hại não (Code)

7. Viết 1 hàm swap 2 số, không dùng biến tạm.

8. Tìm số lớn hơn trong 2 số đưa vào. **Không được dùng if/else hoặc các phép so sánh = < >**.

Các bạn có giải được câu nào không? Nếu không được cũng đừng buồn, hãy xem đáp án ở dưới nhé

1. Thuật toán khá đơn giản, chắc bạn nào cũng nghĩ ra. Ta sử dụng 2 con trỏ p1 và p2, con trỏ p2 sẽ nằm sau con trỏ p1 ($n-1$) phần tử. Cho 2 con trỏ này chạy cùng lúc, khi con trỏ p2 chạy tới cuối list, con trỏ p1 sẽ trỏ tới phần tử thứ n từ cuối lên

```
1  LinkedListNode nthToLast(LinkedListNode head, int n) {
2      if (head == null || n < 1) {
3          return null;
4      }
5
6      LinkedListNode p1 = head;
7      LinkedListNode p2 = head;
8      for (int j = 0; j < n - 1; ++j) { // p2 chạy hơn p1 n-1 phần tử
9          if (p2 == null) return null; // Không tìm ra vì mang ít hơn n phần tử
10         p2 = p2.next;
11     }
12
13     //Cho 2 con trỏ cùng chạy tới cuối list
14     while (p2.next != null) {
15         p1 = p1.next;
16         p2 = p2.next;
17     }
18     return p1;
19 }
```

2. Một câu hỏi khá đơn giản nữa. Chỉ việc **lấy data của node ngay sau note hiện tại, sau đó trả tới note tiếp sau** là xong:

1. Lấy data của note sau a->b->c->d->e sẽ thành a->b->d->d->e
2. Trả tới node tiếp sau: a->b->d->e

Điểm lắt léo của câu đó này là: **Nếu như node cần xóa là node cuối cùng của list, ta không thể xóa được**, chỉ có thể set data của nó là null. Bạn cần phải chỉ ra được điều này thì mới xem như trả lời đúng.

```
1 public static boolean deleteNode(LinkedListNode n) {  
2     if (n == null || n.next == null) {  
3         return false; // Failure  
4     }  
5     LinkedListNode next = n.next;  
6     n.data = next.data;  
7     n.next = next.next;  
8     return true;  
9 }
```

3. Một bài toán đó khá thú vị, bạn nào từng làm việc với bit chỉ cần nhìn là đoán ra ngay. Toán tử & tức là áp dụng toán tử AND từng bit của 2 số. ($n \& (n-1)$) == 0 nghĩ là bit biểu diễn 2 số này không có số 1 nào chung. Có thể xem 2 trường hợp sau:

11001100 và 00110011

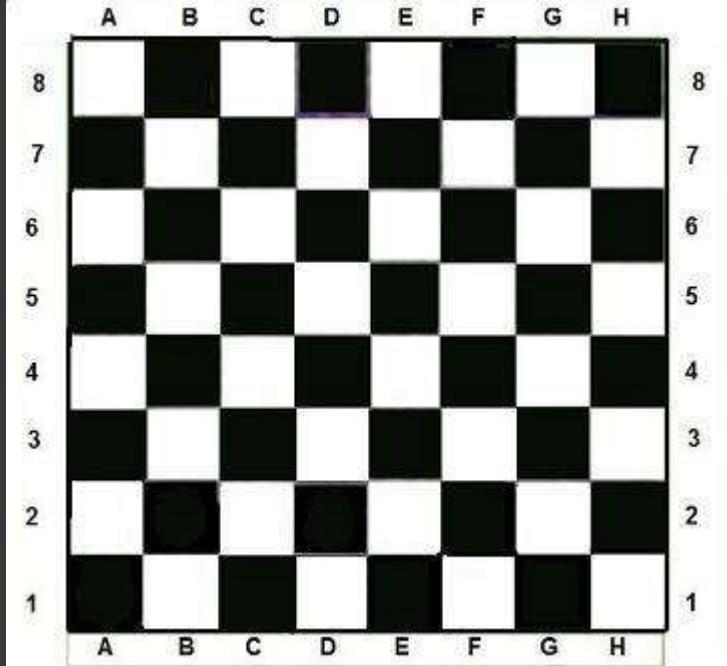
$100000000 (n)$ và $01111111 (n-1)$

Dễ thấy ở trường hợp 2, 2 số n và n-1 không có số 1 chung nào. Vậy n sẽ là 1 số mà các bit của nó là: 10, 100, 1000, 10000. Các bạn gần đoán ra rồi đấy, nếu đổi từ nhị phân sang thập phân, n sẽ là: 2, 4, 8, 16 ... Kết luận: **Dòng code có tác dụng xác định giá trị n là một số lũy thừa của 2.**

4. Bài này bạn có thể giải bằng cách mò và đoán, tuy nhiên người phỏng vấn muốn thử cách suy luận logic của bạn. Đầu tiên, hãy thử phân tích xem số 8 có thể được tạo thành từ các số nào:

1. $8 = 1 * 8 = 1 + 7 = 2 * 4 = 4 + 4 = 4 * 2 \dots$
 2. Tách tiếp ra, ta sẽ thấy: $4 = 3 + 1$ (2 số đầu), $2 = 6 / 3$ (2 số cuối).
 3. Ta đảo vị trí 3 và 6 để ra đáp án cuối cùng: $(3 + 1) / 3 * 6 = 8$.
 4. Còn 1 đáp án trên trời rơi xuống nữa là: $3 - 1^3 + 6 = 8$ nhé.
5. Một bài toán suy luận khá hay. Hãy thử đặt 1 quân lên bàn cờ, quân domino chiếm 2 ô liền nhau, 1 đen 1 trắng. Ta có thể suy luận: Mỗi quân domino chiếm 1 ô đen và 1

ô trắng, 31 quân sẽ chiếm 31 ô đen, 31 ô trắng. Do bàn cờ đã bị mất 1 ô góc trên bên trái, 1 ô góc dưới bên phải, 2 ô này cùng màu với nhau (Bạn nhìn hình sẽ thấy). Do đó bàn cờ chỉ còn 32 ô đen, 30 ô trắng, **không thể dùng 31 quân domino để xếp đầy bàn cờ**.



6. Câu này không khó, chỉ tương đương với 1 bài trắc nghiệm trong trường. **Code sẽ không chạy vì $i = 100$, không thỏa điều kiện $i \leq 0$** . Câu này chỉ thử khả năng suy luận và đọc code của bạn thôi.

7. Một bài đố mèo đơn giản thường được các thầy dùng để số sinh viên

```

1  public static void swap(int a, int b) {
2      //b = 9, a = 5
3      a = b - a; // 9 - 5 = 4
4      b = b - a; // 9 - 4 = 5
5      a = a + b; // 4 + 5 = 9
6      //b = 5, a = 9
7  }
8
9  //Một cách khác optimize và nhanh hơn, sử dụng toán tử XOR
10 public static void swap_opt(int a, int b) {
11     a = a^b;
12     b = a^b;
13     a = a^b;
14 }
```

8. Bài toán này khá khó, ta có thể giả bằng cách “suy luận ngược” như sau:

1. Nếu $a < b$: return b , ngược lại return a
2. Nếu $a - b < 0$: return b , ngược lại return a
3. Ta nhầm:
 $b = a - (a - b) = a - 1*(a-b) = a - k*(a-b)$ với $k = 1$
 $a = a - 0 = a - 0*(a-b) = a - k*(a-b)$ với $k = 0$
4. Kết hợp 2 và 3 ta có:
5. Return $a - k*(a-b)$. Nếu $a - b < 0$: $k = 1$, ngược lại $k = 0$
6. Gọi $c = a-b$. Ta không thể so sánh c với 0, tuy nhiên ta biết rằng $c < 0$ nếu c là số âm. Số âm có đặc tính gì nhỉ: **bit ngoài cùng bên trái của nó sẽ bằng 1. Bit ngoài cùng bên trái của số dương sẽ bằng 0.** Ô, đúng là số k chúng ta đang tìm còn gì.

Lời giải khá ngắn gọn, nhưng logic để có được lời giải này cũng *không phải dạng vừa đâu*.

```

1  int getMax(int a, int b) {
2      int c = a - b;
3      int k = (c << 31) & 0x1;
4      int max = a - k * c;
5      return max;
6  }
```

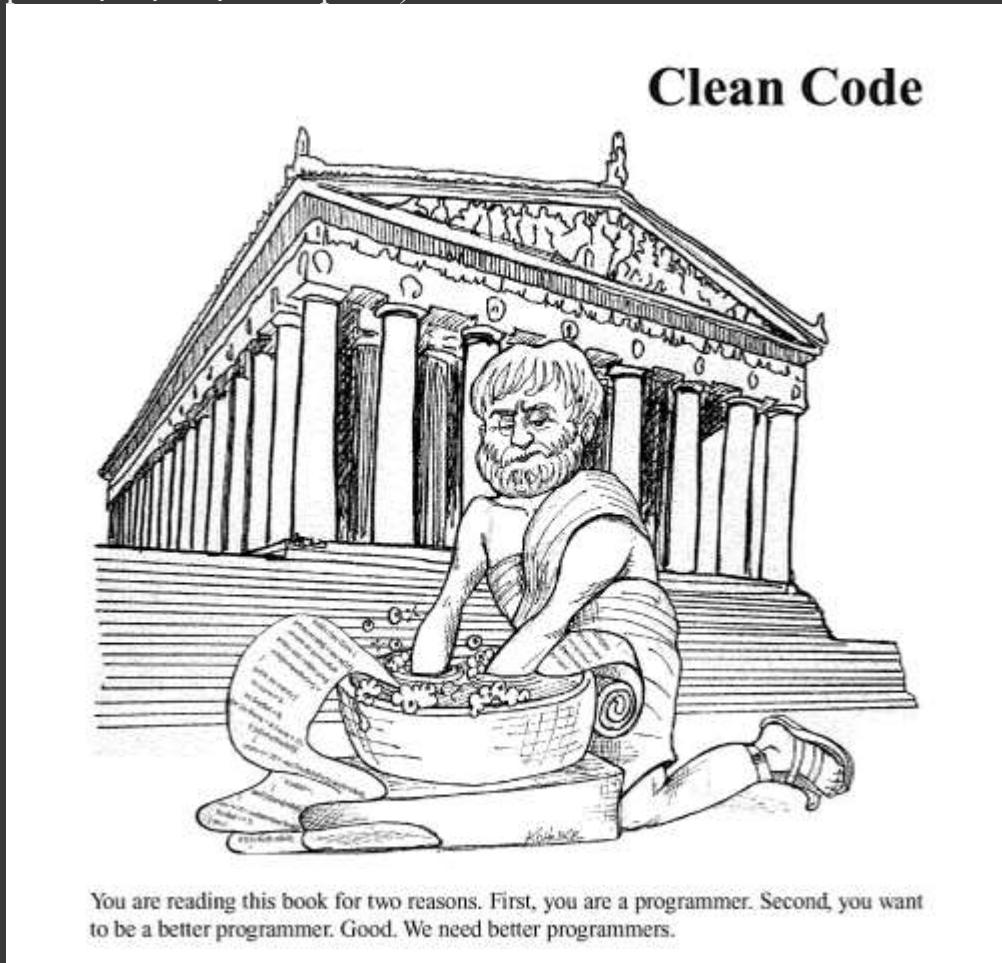
Tới đây mình chắc các bạn cũng khá nhức não rồi, mình cũng vậy. Lâu lâu hãy đọc những câu hỏi như thế này để tăng tư duy lập trình nhé. Hẹn gặp lại các bạn ở bài viết sau.

Luận về comment code (Phong cách kiểm hiệp)

Posted on 09/07/2015 by Pham Huy Hoàng

Comment code luôn là vấn đề gây tranh cãi sút đầu mẻ trán trong giới võ lâm. Xưa kia, thuở còn mài đít trên ghê nhà trường, ta thường được các thầy dặn rằng: Code nhớ phải comment. Thuở mới đi làm, sơ nhập gian hò, mỗi khi đọc code không hiểu, ta cũng hay đập bàn mà chửi: “Thằng này code ngu quá đ*o comment gì cả”.

Thế nhưng, lưu lạc giang hồ bao năm, đọc code cũng nhiều; từ code không comment cho tới code comment đầy đủ và sạch sẽ; ta vẫn băn khoăn không biết thật sự phải comment thế nào mới đúng. Thế rồi, sau khi đọc Clean Code, ta như lượm được bí tịch võ công trong truyền thuyết. Ta ngộ ra được cái gọi là “đạo code”, “đạo comment”, đặt 1 bước chân vào con đường võ đạo đỉnh cao (Đạo ở đây là đạo lý, ko phải đạo tặc trộm cướp nhé).



Thần Điêu Đại Hiệp của Kim Dung có đoạn: Dương Quá bị lạc vào kiếm mộ của Độc Cô Cầu Bai. Thuở còn sống, Độc Cô cầu bại là người kiếm thuật vô song, danh chấn thiên hạ. Trước khi chết, ông chôn 4 thanh kiếm ở nơi gọi là kiếm mộ. Quý đạo hữu có thể đọc cốt sự đó ở đây: [Độc Cô Cầu Bai](#). Tại hạ muôn quý vị chú ý chi tiết này: *Một hồi sau, chàng mới đặt thanh kiếm nặng đó xuống, nhác thanh kiếm thứ ba lên, lần này chàng lại bị lầm. Chàng cứ tưởng thanh kiếm này phải nặng hơn thanh kiếm vừa rồi, nên vận lực ra cánh tay. Nào ngờ nó nhẹ tênh như không, chàng ngưng thần xem kỹ, hóa ra đó là một thanh kiếm gỗ, chôn dưới đá lâu năm, thân và cán kiếm đều đã bị mục, đọc dưới mặt đá có khắc dòng chữ:*

Sau bốn mươi tuổi, không mang binh khí,

Thảo mộc trúc thạch đều có thể dùng làm kiếm.

Cứ thế tinh tu, đạt tới cảnh giới vô kiếm thắng hữu kiếm.

Tổng kết lại, cuộc đòi tu kiếm của Độc Cô Cầu Bai bao gồm ba giai đoạn:

- Lúc trai trẻ lòng đầy nhiệt huyết mà thiếu sự chín chắn thì sử dụng tử vi kiếm là thanh bảo kiếm sắc, nhẹ và linh hoạt.
- Khi đạt độ chín của suy nghĩ và sức lực, sử dụng thanh kiếm sắt nặng nề mà không sắc bén.
- Khi bắt đầu về già, suy nghĩ và kiếm thuật đạt trình độ cao, vũ khí chỉ còn là thanh kiếm gỗ và đạt mức thượng thừa thì không kiếm mà thắng đối thủ, bất cứ thứ gì cũng là kiếm.

Ta tự thấy hầu như **lập trình viên nào cũng sẽ phải trải qua ba giai đoạn này** trên con đường “cầu đạo”. Kẻ nào ngộ tính cao thì chỉ cần làm việc một hai năm đã đạt tới

cảnh giới cuối. Ké nào đầu óc u mê lạc lối thì cả đời vẫn cứ ở giai đoạn hai mà thôi.



Giai đoạn trẻ trâu đầy nhiệt huyết (Code không comment hoặc comment lung tung)

Ké nào sơ nhập giang hồ đều trải qua giai đoạn này. Đây là khi ta còn ngồi trên ghế nhà trường, hoặc mới đi làm. Code chạy được là mừng, đôi khi code cho xong là nộp, chả cần comment. Nhiều khi ta nghe lời các sư phụ, thêm comment vào code. Đống comment này nhiều nhưng khá vô dụng, đọc rất buồn cười, như thể mấy kẻ mới học võ công mà bày trò múa máy bắt chước các chiêu thức cao siêu trong võ học.

```
1 //Chia đôi toàn bộ các phần tử của mảng đưa vào
2 public int[] half(int[] y){
3     int[] x; //Tạo một array mới chứa kết quả
4     for (int i = 0; i < y.length ; i++) //Duyệt các phần tử của mảng y
5     {
6         x[i] = y[i]/2; //Gán giá trị vào array chứa kết quả
7     }
8     return x; //Trả kết quả ra
9 }
```

Ta thấy code tuy comment khá đầy đủ, nhưng chẳng có ý nghĩa mấy, thời gian viết comment còn nhiều hơn viết code. Một số coder thường dùng trò: Thêm comment để biết ngày giờ sửa code, comment lại 1 số code không dùng nữa. Xin lỗi, cho phép tại ha nói thẳng:

- Code không dùng nữa thì xóa đi, đừng comment. Có SVN rồi khi cần code cũ chỉ việc restore lại là xong.
- Đừng comment theo kiểu: //02/03/1992 Sửa tên class. Sử dụng SVN có thể tra ra log rõ ràng và tiện dụng hơn nhiều.

Giai đoạn tạm chín chắn trong suy nghĩ (Biết cách comment)

Code một thời gian, hầu như mọi ltv sẽ đạt đến giai đoạn này. Ở giai đoạn này, ta đã cảm thụ được một chân lý vô học : **Comment nên là what, không phải how**. Comment để nói code làm gì, không phải để giải thích việc code làm. Lúc này, lập trình viên đã hiểu rõ hơn về cách comment. Với những đoạn code phức tạp, dài dòng, 1 dòng comment ngắn gọn sẽ giúp người sau dễ dàng hiểu đoạn code làm gì

```
1 public object DoThing() {  
2     //Lấy 1 phần tử random từ database  
3     //Một đống code phức tạp  
4     return obj;  
5 }
```

Tuy vậy, các bậc cao nhân xưa đã rút ra được một điều: **Comment là thứ dối trá**. Khi sửa code, comment thường không được sửa, sẽ dẫn tới tình trạng: code một đằng, comment một néo. Ta phải đọc cả code lẫn comment để biết code làm gì, phí gấp đôi thời gian. **Cách comment tốt nhất chính là dùng code, chính code sẽ nói code làm gì**. Ngô ra được điều này, các đạo hữu sẽ đạt tới cảnh giới cuối cùng trong “code học”. Cảnh giới vô kiểm thắng hữu kiểm (Vô comment thắng hữu comment)

Đây là cảnh giới mà ta hy vọng các đạo hữu có thể đạt được

Ở cảnh giới này, ta code không cần comment nhiều. Giống như Độc Cô Cầu Bá có thể lấy kiếm gỗ làm kiếm, lấy lá cỏ làm kiếm, lấy tay làm kiếm; ta thấy bất cứ thứ gì cũng có thể làm comment: tên biến cũng là comment, tên hàm cũng là comment, tên database cũng là comment. Code cũng như comment, comment cũng như code, code và comment tuy hai mà một.

Tuy nhiên, đôi khi **bắt buộc phải dùng comment**: Khi viết JavaDoc, API v...v, ta bắt buộc phải comment và document, vì người dùng API chỉ được đọc document chứ không đọc code. Một số trường hợp khác: **comment là why chứ không phải what**. Ta viết comment để người khác (hoặc ta sau này) biết **vì sao ta lại viết code như vậy**. Comment có tác dụng **nói những điều bẩn thân code không nói được**. Còn việc code làm gì, chạy như thế nào, chỉ cần đọc code là hiểu

```
1 //Chỉ cần nhìn tên hàm là biết hàm làm gì  
2 public object GetRandomObjectFromDatabase() {  
3     return randomObj;  
4 }
```

```
5 public int[] HalfAllNumbersFromInput(int[] input){  
6     int[] output; //Nhìn tên biến là biết biến làm gì  
7     for (int i = 0; i < input.length ; i++)  
    {  
8         output[i] = input[i]/2;  
9         //Viết để debug, sau này phải remove  
10        //Đây là comment bắt buộc phải viết, để giải thích lý do ta viết code  
11        Console.WriteLine("Call this");  
12    };  
13    return output;  
14 }  
15  
16 }
```

Ở đây, ta không phủ nhận độ hữu dụng của comment. Nhưng vấn đề là: **Nhiều gã coder lợi dụng comment để viết code không rõ ràng, khó hiểu, sau đó sử dụng comment để lấp liếm**. Điều ta muốn các bằng hữu hiểu là: Hãy có **gắng viết code một cách rõ ràng nhất có thể**, bằng cách đặt tên biến, tên hàm, tách code, ... Đó mới là cảnh giới tối cao của “code đạo”. Đừng học đòi theo lũ trẻ trâu mà viết code kiểu “càng ngắn càng tốt”, càng khó hiểu càng tốt, đó chỉ là cái dũng của kẻ thất phu mà thôi. Chúc các bằng hữu sớm thành cao thủ trên con đường cầu đạo, nhầm, cầu code của mình.

[Tutorial] Tạo ứng dụng chat với 50 dòng code, Firebase và AngularJS

Posted on 07/07/2015 by Phạm Huy Hoàng

Từ lúc viết blog tới giờ, mình chưa có bài nào hướng dẫn các bạn tạo ra một sản phẩm từ đầu tới cuối cả. Đến hôm nay nhìn lại, lương tâm cắn rứt nên mình sẽ hướng dẫn các bạn viết một thứ thật **hoàng tráng**. Nghĩ đi nghĩ lại, tính mình vốn lười, viết dài thì rất mất thời gian, do đó mình sẽ tìm cách code ít nhất, cho ra sản phẩm ảo nhất. Và đó là lý do bài viết này ra đời: **Một ứng dụng chat chỉ với 50 dòng code**.

Một số kiến thức cơ bản bạn cần biết để thực hiện bài tutorial này. Bạn có thể code với **Notepad hoặc Notepad++, không cần IDE** gì cả:

1. HTML
2. Javascript
3. AngularJS (Chỉ cần ở mức độ Beginner, biết một chút thôi, không cần nhiều).

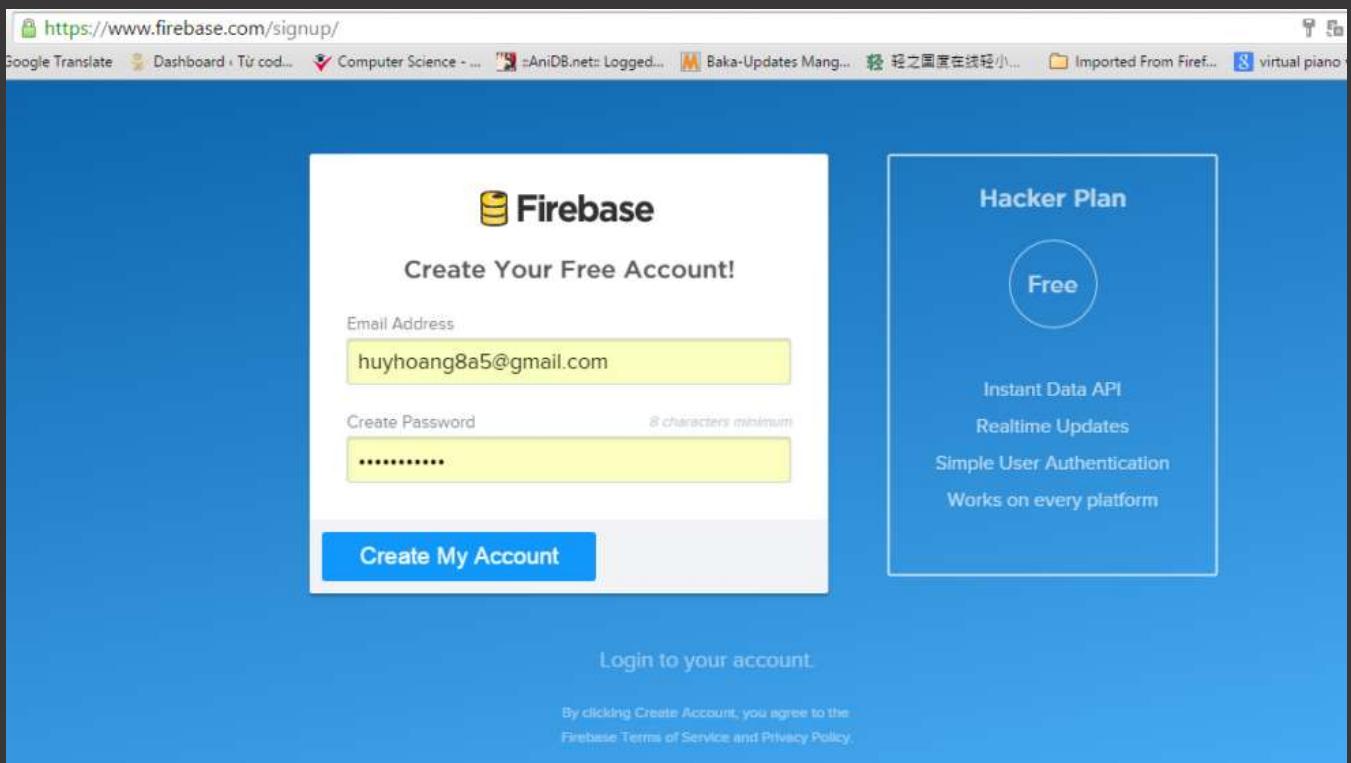
Hãy đọc bài viết này, bạn đã có đủ kiến thức cơ bản AngularJS để làm theo tutorial: <http://sangplus.com/angular-js-101-gioi-thieu-ve-angularjs.html>

Đầu tiên, mình sẽ mô tả về ứng dụng chúng ta sẽ làm:

1. Khi người dùng vào ứng dụng, người dùng được yêu cầu nhập tên.
2. Sau khi nhập tên, người dùng nhập nội dung vào textbox, bấm enter. Mọi người khác trong phòng chat sẽ thấy tin nhắn mới.

Một số bạn sẽ hỏi: Không có database để lưu dữ liệu, không có server để xử lý thì làm sao chat, làm sao realtime như chat được. Để đơn giản, tất cả những việc như: **Lưu trữ dữ liệu, đồng bộ dữ liệu trên client sẽ được xử lý bởi Firebase**, một dịch vụ cloud hỗ trợ xây dựng ứng dụng realtime.

Các bạn hãy vào trang web của Firebase <https://www.firebaseio.com/signup/>, và đăng ký một tài khoản mới:



Sau khi login, các bạn sẽ thấy giao diện sau. Điền thông tin, bấm create new app để tạo 1 app Hãy nhớ url của app này nhé, bạn sẽ cần đấy.

The screenshot shows the Firebase Dashboard with two apps listed: 'CHATTING-SAMPLE' and 'MY FIRST APP'. Both apps are in 'DEVELOPMENT ONLY' mode under the 'HACKER PLAN'. Each app card shows its name, URL (chatting-sample.firebaseio.com and amber-torch-2590.firebaseio.com respectively), and a 'Set up Hosting' button. Below the cards are 'Manage App' and 'Upgrade Plan' buttons, along with 'Add a developer' links. On the left, there's a sidebar with 'APP NAME' and 'APP URL' fields, and a 'CREATE NEW APP' button. A green notification bar at the top right says 'Firebase created successfully!'. The main message on the dashboard reads: 'Your account has been created successfully! Take a look at these resources to help you get started.'

Sau khi có URL để kết nối tới service của Firebase, chúng ta có thể bắt đầu code. Mở Notepad, Notepad++ hay IDE của bạn, sau đó tạo 1 file html trống nhé.

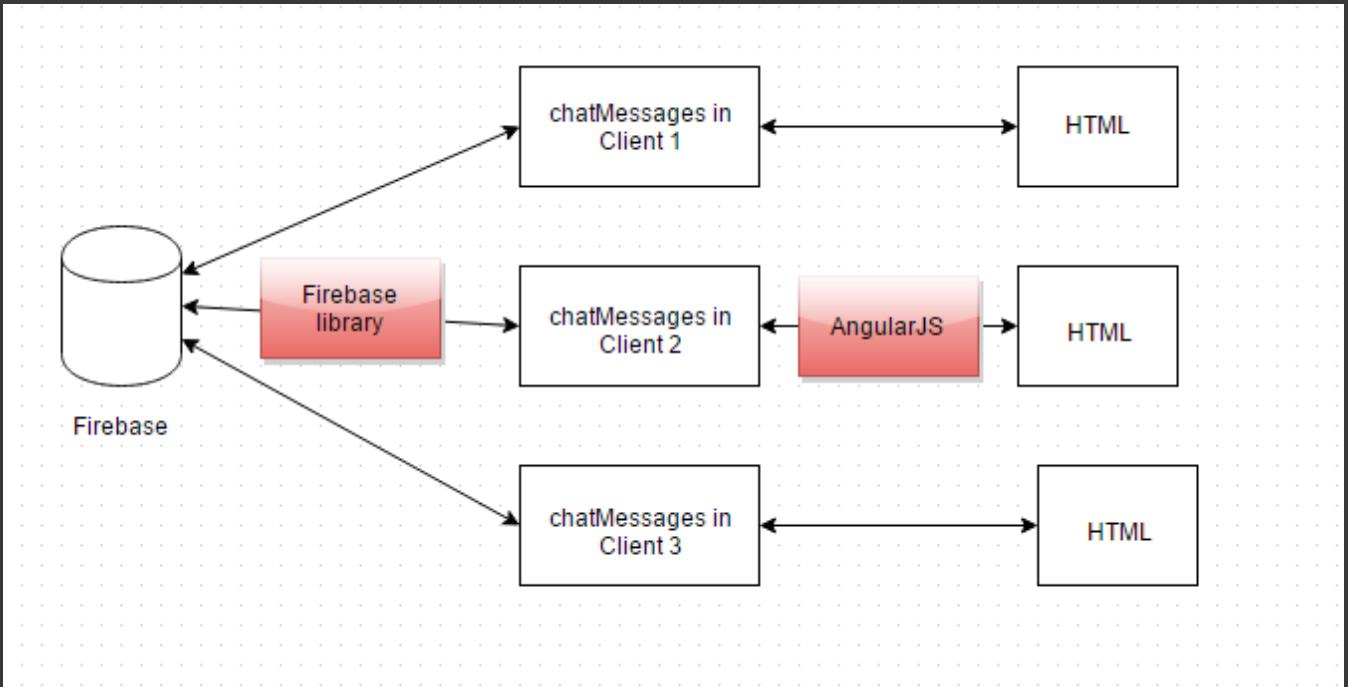
Ta thêm 3 file script, trở tới 3 thư viện cần dùng, đó là AngularJS, Firebase, và AngularFire (Thư viện hỗ trợ Firebase cho AngularJS).

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Firebase App</title>
5     <script
6       src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.2/angular.min.js"></script>
7     <script src="https://cdn.firebaseio.com/js/client/2.0.4.firebaseio.js"></script>
8     <script
9       src="https://cdn.firebaseio.com/libs/angularfire/0.9.0/angularfire.min.js"></script>
10    </head><!-- AngularJS -->
11    <!-- Firebase -->
12    <!-- AngularFire -->
13
14  <body>
15  </body>
16  </html>
```

Trước khi viết html, chúng ta hãy viết code javascript trước, với một số hàm để lấy và gửi dữ liệu như sau:

```
1 var app = angular.module('app', ['firebase']); //Tạo 1 module của angular,
2 inject module firebase vào
3 app.controller('chatCtrl', ['$scope', '$firebase',
4   function($scope, $firebase) {
5     var name = prompt("Enter your name: ", '');
6     $scope.name = name; //Lấy tên của người dùng
7     $scope.chatMessage = ""; //Tẩy trắng khung text
8     //Kết nối tới service của firebase, url ở đây là url app của bạn ở
9     bước trên nhé
10    var ref = new Firebase(
11      "https://amber-torch-****.firebaseio.com/");
12    var sync = $firebase(ref);
13    $scope.chatMessages = sync.$asArray(); //Lấy toàn bộ dữ liệu trong
14    database trên Firebase, biến nó thành 1 array các object trong javascript
15    $scope.sendChat = function() {
16      var chatMessage = {
17        name: name,
18        message: $scope.chatMes
19      };
20      $scope.chatMessages.$add(chatMessage); //Thêm 1 tin nhắn vào
21      array
22      $scope.chatMes = "";
23    }
24  ]);
25];
```

Code có vẻ cũng khá đơn giản nhỉ? Cơ chế hoạt động của chương trình như sau



1. Toàn bộ dữ liệu được lưu trữ ở database của firebase. Dữ liệu sẽ được đồng bộ 2 chiều xuống array chatMessages ở clients, thông qua thư viện \$firebase.
2. Dữ liệu trong array chatMessages sẽ được chuyển thành html, đồng bộ 2 chiều nhờ AngularJS.
3. Ở client, khi chúng ta thêm 1 tin nhắn vào array chatMessages, dữ liệu sẽ được đồng bộ ngược lên firebase, sau đó đồng bộ từ firebase sang các client khác.

Do việc đồng bộ từ client => Firebase => client khác đã được thư viện lo, chúng ta chỉ viết phần đồng bộ từ client lên html. Bắt đầu viết html nào

```

1 <body>
2   <h1>Firebase chatbox</h1>
3
4   <div>
5     <div id="chatBox" style="padding: 10px; border: black 1px solid">
6       <h1>Chat box</h1>
7
8         <!-- Đồng bộ từ array chatMessage sang HTML.
9           Hàm limit To để lấy 15 messages cuối cùng -->
10        <div>
11          <span style="font-weight:
12            bold">{{chatMessage.name}} : 
13            <span>{{chatMessage.message}} </span>
14        </div>
15      </div>
16
17      <form> <pre> Name: {{name}}</pre>
18      <!-- Khi người dùng bấm enter, ta gọi hàm sendChat trong controller,
19      -->
```

```
17         thực hiện đồng bộ, hiển thị tin nhắn trên máy những người dùng  
18     khác -->  
19         Chat: <input type="text" ng-model="chatMes" />  
20             <button type="submit" ng-click="sendChat()">Send</button>  
21     </div>  
22 </body>  
23
```

Nếu thực hiện đúng, chương trình sẽ có giao diện như sau:

Firebase chatbox

Chat box

Hoang : hê lô
Hoang : hello
Hoang : hello2
Hoang : fdfsffsd
Ho : 445
Ho : 454
Ho : 4243423
Ho : 32423432
Ho : 32423342
Ho : 32423432
Ho : 32423432

Name: Chat: Send

Sau khi chat thử, bạn vào url trên database sẽ thấy data như sau:

```

    amber-torch-2590
    +-- ~JqSjJN0eBmAjq88Vq3o
        |   message: "hê lô"
        |   name: "Hoang"
    +-- ~JrwgpiCYwdTcUNcEWjo
        |   message: "hello"
        |   name: "Hoang"
    +-- ~JrwgqBheveOB0rff63N
    +-- ~JrwgsaHgK6gYcjCBmlv
    +-- ~JrwauHeTQxmr3vhb_ KR

```

Tổng hợp toàn bộ code của chương trình, đúng 52 dòng nhé

```

1 <html xmlns="http://www.w3.org/1999/xhtml">
2
3     <head>
4         <title>Firebase App</title>
5     </head>
6     <!-- AngularJS -->
7     <script
8         src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.2/angular.min.js"></script>
9     <!-- Firebase -->
10    <script src="https://cdn.firebaseio.com/js/client/2.0.4.firebaseio.js"></script>
11    <!-- AngularFire -->
12    <script
13        src="https://cdn.firebaseio.com/libs/angularfire/0.9.0/angularfire.min.js"></script>
14
15    <body ng-app="app">
16        <h1>Firebase chatbox</h1>
17        <div ng-controller="chatCtrl">
18            <div id="chatBox" style="padding: 10px; border: black 1px solid">
19                <h1>Chat box</h1>
20                <div ng-repeat="chatMessage in chatMessages | limitTo:-15"> <span
21                    style="font-weight: bold">{{chatMessage.name}}</span> :
22                    <span>{{chatMessage.message}}</span>
23                </div>
24            </div>
25            <form> Name: {{name}} Chat:

```

```
23          <input type="text" ng-model="chatMessage" />
24          <button type="submit" ng-click="sendChat()">Send</button>
25      </form>
26  </div>
27 </body>
28 <script>
29 var app = angular.module('app', ['firebase']);
30 app.controller('chatCtrl', ['$scope', '$firebase', function($scope, $firebase) {
31     var name = prompt("Enter your name: ", '');
32     $scope.name = name;
33     $scope.chatMessage = "";
34     var ref = new Firebase("https://amber-torch-2590.firebaseio.com/");
35     var sync = $firebase(ref);
36     $scope.chatMessages = sync.$asArray();
37     $scope.sendChat = function() {
38         var chatMessage = {
39             name: name,
40             message: $scope.chatMes
41         };
42         $scope.chatMessages.$add(chatMessage);
43         $scope.chatMes = "";
44     }
45     $scope.clear = function() {
46         for(var i = 0; i < $scope.chatMessages.length; i++) {
47             $scope.chatMessages.$remove($scope.chatMessages[i]);
48         }
49     }
50 });
51 </script>
52 </html>
```

Các bạn có thể chạy thử bản online của ứng dụng tại
đây:<http://conanak99.github.io/chatBox.html>. Đây là bài tutorial đầu tiên của mình.
Mình sẽ cập nhật một số bài khác nếu các bạn ủng hộ :D.

Giải thích đơn giản về CI – Continuous Integration (Tích hợp liên tục)

Posted on 27/08/2015 by Phạm Huy Hoàng

Với các bạn sinh viên, khái niệm Continuous Integration (Tích hợp liên tục) là một cái gì đó nghe rất cao siêu và hoành tráng. Mình sẽ nêu khái niệm, sau đó đưa ra một câu chuyện đơn giản để giải thích cho khái niệm này.

Tích hợp liên tục (CI) là phương pháp phát triển phần mềm đòi hỏi các thành viên trong nhóm tích hợp công việc thường xuyên. Mỗi ngày, các thành viên đều phải theo dõi và phát triển công việc của họ ít nhất một lần. Việc này sẽ được một nhóm khác kiểm tra tự động, nhóm này sẽ tiến hành kiểm thử truy hồi để phát hiện lỗi nhanh nhất có thể. Cả nhóm thấy rằng phương pháp tiếp cận này giúp giảm bớt vấn đề về tích hợp hơn và cho phép phát triển phần mềm gắn kết nhanh hơn. Trích từ: <http://www.ibm.com/developerworks/vn/library/rational/201301/continuous-integration-agile-development/>



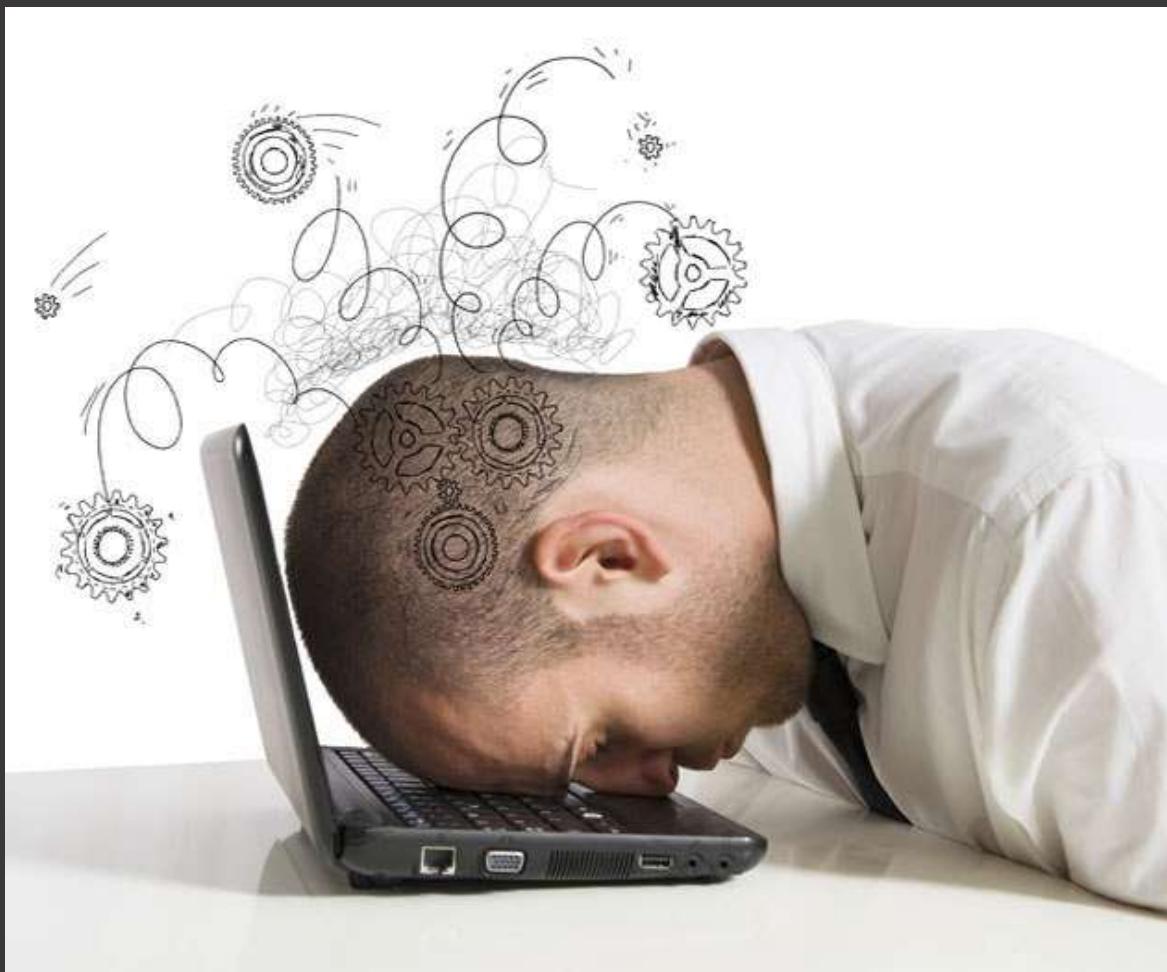
Nếu không hiểu ảnh nói gì, hãy đọc câu chuyện nho nhỏ phía dưới

Ngày xưa ngày xưa, Nam còn là sinh viên ngành IT của trường B. Mỗi lần code, làm bài tập nhóm đối với Nam là một cực hình. Cả team ngồi phác thảo ra từng module nho nhỏ, sau đó chia ra, mạnh ai về nhà code. Cuối tuần, cả nhóm hẹn nhau ra quán cafe để “ráp code”, kop các phần đã làm qua USB, bỏ vào một project chung. Mỗi lần “ráp code”, chương trình không build được, một núi lỗi xuất hiện, cả nhóm phải hì hục mất nguyên buổi chiều để sửa. “Ráp code” trở thành một cơn ác mộng đối với Nam và các bạn trong nhóm.

Ra trường rồi đi làm, Nam đi phỏng vấn xin việc, được nhận vào công ty phần mềm C. Ở đây, mọi người đã biết dùng SVN nên không còn cảnh phải copy code qua lại nữa. Từ “ráp code” cũng biến mất mà thay vào đó là từ “tích hợp”. Tuy nhiên, team

của Nam vẫn còn làm việc khá ầu. Mỗi sáng, các thành viên trong team update code từ SVN/[Git](#)về, code say sưa, sau đó commit code lên trước khi về nhà.

Đôi khi code không build được, cả team lại nháo nhào “truy tìm thủ phạm”: anh Phạm Văn A sửa code mà quên commit file mới lên, chị Lê Thị B sửa connection string, ... Đời nhiều khi còn trái ngang hơn, anh A sửa code, chị B sửa code làm phần của Nam chạy bị lỗi, thế là Nam lanh đú. Mỗi cuối tuần, cả team lại phải OT để tích hợp và sửa lỗi.

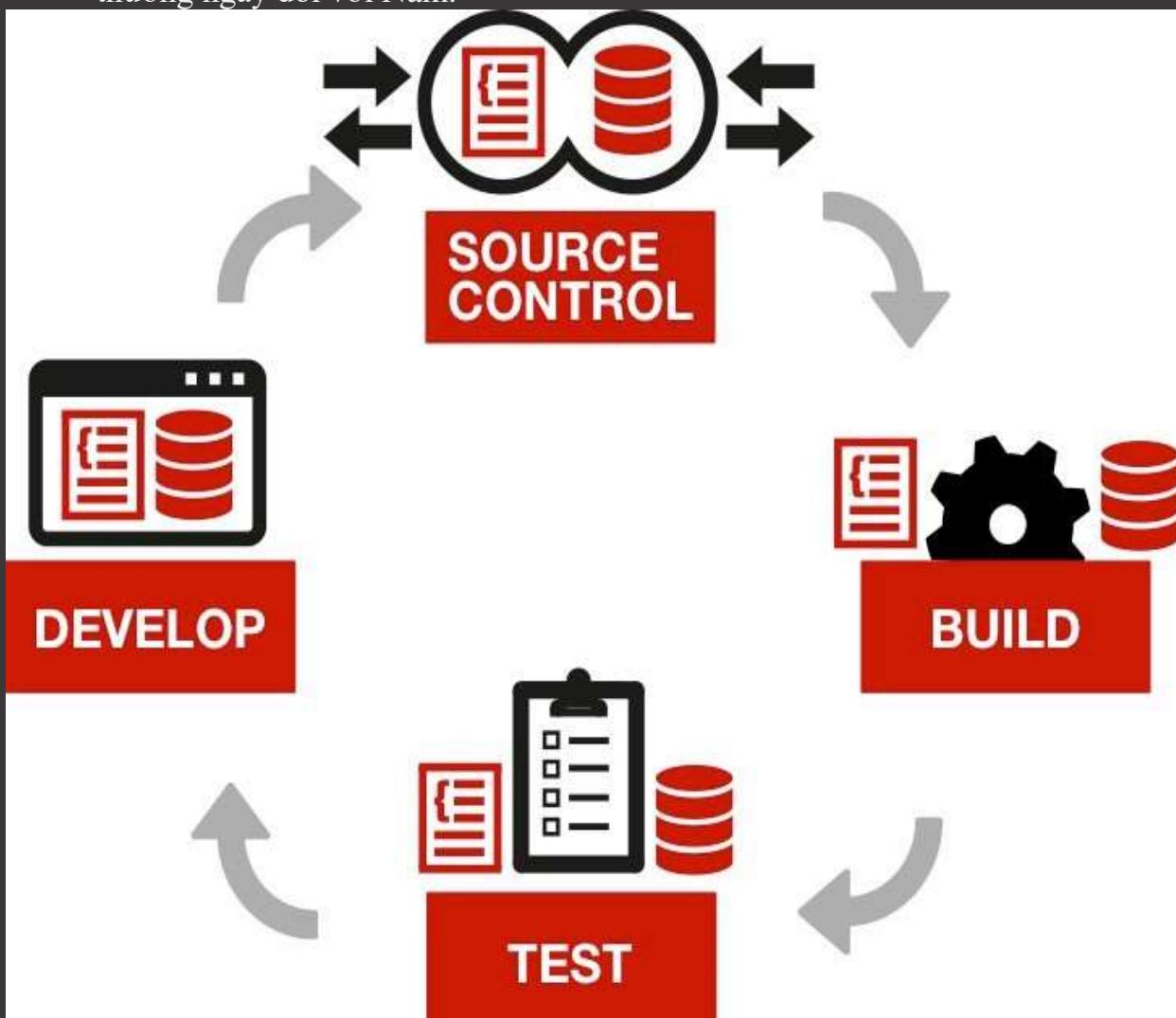


Phần mềm không bán được vì quá nhiều lỗi, release chậm, công ty cũ phá sản, Nam phải đi tìm việc mới. Với khả năng của mình, Nam được nhận vào công ty C. Công ty này khá chuyên nghiệp, có áp dụng Agile, CI – tích hợp liên tục. Nhờ có CI, team của Nam hiện tại không còn gặp những rắc rối khi tích hợp:

- Mỗi khi có người commit code, hệ thống CI sẽ tự lấy code từ SVN, thực hiện build. Hệ thống sẽ gửi mail thông báo cho toàn bộ thành viên nếu như build bị

lỗi. Cả nhóm chỉ việc đọc mail, xem ai là người commit revision đó, và “nắm đầu” thủ phạm, bắt hắn sửa lỗi.

- Project của Nam được viết Unit Test rất cẩn thận, đầy đủ. Khi anh A, chị B sửa code, commit lên, hệ thống sẽ build và chạy lại toàn bộ các Unit Test. Nếu có case nào bị fail, cả team sẽ nhận được thông báo, và anh A, chị B phải nhanh chóng fix code để các case này pass.
- Việc tích hợp được diễn ra hàng ngày, nhiều lần trong ngày. Mỗi khi ai đó commit code làm hư build hoặc gây lỗi, cả team có thể giải quyết vấn đề NGAY LẬP TỨC. “Tích hợp” không còn là nỗi ác mộng mà trở thành chuyện thường ngày đối với Nam.



Nói tóm lại, CI – tích hợp liên tục là phương pháp mà các team Agile sử dụng để đảm bảo code của toàn dự án luôn build được, luôn chạy đúng (Pass toàn bộ các test case). Hiện tại, hầu như các công ty phần mềm đều áp dụng CI, thông qua một số framework

như: TFS, TeamCity, Hudson, Jenkin, Travis, ... Tự trang bị cho mình những kiến thức về CI sẽ khiến bạn có giá hơn trong mắt nhà tuyển dụng nhé.

Nếu muốn hiểu chuyên sâu về Continous Integration và áp dụng, bạn nên xem một bài viết chi tiết. cụ thể về Continous Integration của Martin

Fowler: <http://www.martinfowler.com/articles/continuousIntegration.html>. Ở bài tiếp theo, mình sẽ hướng dẫn các bạn cách áp dụng “tích hợp liên tục” vào project, thông qua Travis-CI và github.

[Tutorial] Viết Unit Test trong C# với NUnit

Posted on 25/08/2015 by Phạm Huy Hoàng

Giới thiệu tổng quan về Unit Test

Ở trường đại học chắc các bạn đã được học khái niệm về Unit Test trong môn “Kiểm thử chất lượng phần mềm”. Nói một cách dễ hiểu, unit test tức là **code dùng để test code ta đã viết**.

Một số đặc điểm của unit test:

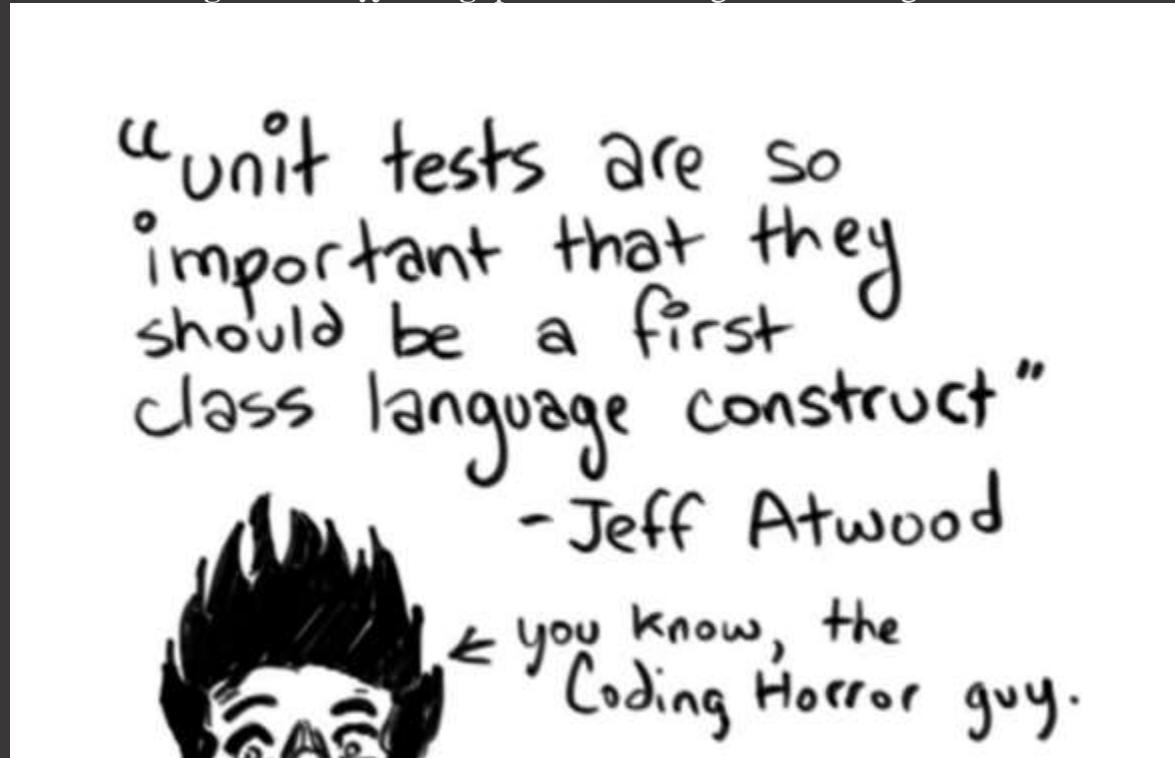
1. Code unit test phải ngắn gọn, dễ hiểu, dễ đọc.
2. Mỗi unit test là 1 đơn vị riêng biệt, độc lập, không phụ thuộc vào unit khác.
3. Mỗi unit test là 1 method trong test class, tên method cũng là tên UnitTest. Do đó ta nên đặt tên hàm rõ ràng, nói rõ unit test này test cái gì (Test_A_Do_B), tên method có thể rất dài cũng không sao.
4. Unit Test phải nhanh, vì nó sẽ được chạy để kiểm định lỗi mỗi lần build. Do đó trong unit test nên hạn chế các task tốn thời gian như gọi I/O, database, network,...
5. Unit Test nên test từng đối tượng riêng biệt. Vd: Unit Test cho Business Class thì chỉ test chính BusinessClass đó, không nên dùng tới các class mớn nối với nó (DataAccess Class chẳng hạn).

Một số bạn sẽ thắc mắc: “Ó hay, trong business class thì phải gọi data access chứ?”, và “code mình dính chùm lắm, làm sao test từng class riêng được”.

- Để viết unit test, các class của bạn phải có quan hệ “lỏng lẻo” với nhau (loose coupling), nếu không viết được unit test nghĩa là các class dính với nhau quá chặt, sẽ khó thay đổi sau này. Hãy áp dụng các nguyên lý **SOLID** vào code, viết code với tư tưởng “viết sao để unit test được” sẽ là code của bạn uyển chuyển, dễ test hơn.
- Về vấn đề Business Class và Data Access, các class không nên gọi trực tiếp lẫn nhau mà gọi thông qua interface. Khi unit test, ta sẽ thay các hiện thực interface này bằng các class giả, thay thế cho class data access thật. Mình sẽ nói rõ về vấn đề này ở những bài viết về **Inversion of Control, Dependency Injection** trong tương lai.

Một số lợi ích của Unit Test:

- Nếu viết Unit Test một cách cẩn thận, **code của bạn sẽ ít lỗi hơn**, vì Unit Test sẽ phát hiện lỗi cho bạn.
- Phát hiện những hàm chạy chậm và không hiệu quả thông qua thời gian chạy của Unit Test.
- Tăng sự tự tin khi code, vì đã có Unit Test phát hiện lỗi.
- Khi refactor code, sửa code hay thêm chức năng mới, **Unit Test đảm bảo chương trình chạy đúng**, phát hiện những lỗi tiềm tàng mà ta bỏ lỡ.



Gần đây, mô hình phát triển TDD (Test Driven Development) đang trở nên hot, được áp dụng nhiều. Mô hình này dựa trên khái niệm: Với mỗi chức năng, ta viết Unit Test trước, sau đó viết hàm hiện thực chức năng để unit test pass. Một số công ty ở Việt Nam cũng đang áp dụng mô hình này, trong khi phỏng vấn xin việc cũng có nhé :D. Các bạn có thể xem thêm về Unit Test và TDD ở đây:<http://www.pcworld.com.vn/articles/cong-nghe/cong-nghe/2005/12/1188434/unit-test-voi-phat-trien-phan-mem-hien-dai/>

Unit Test trong C# với NUnit

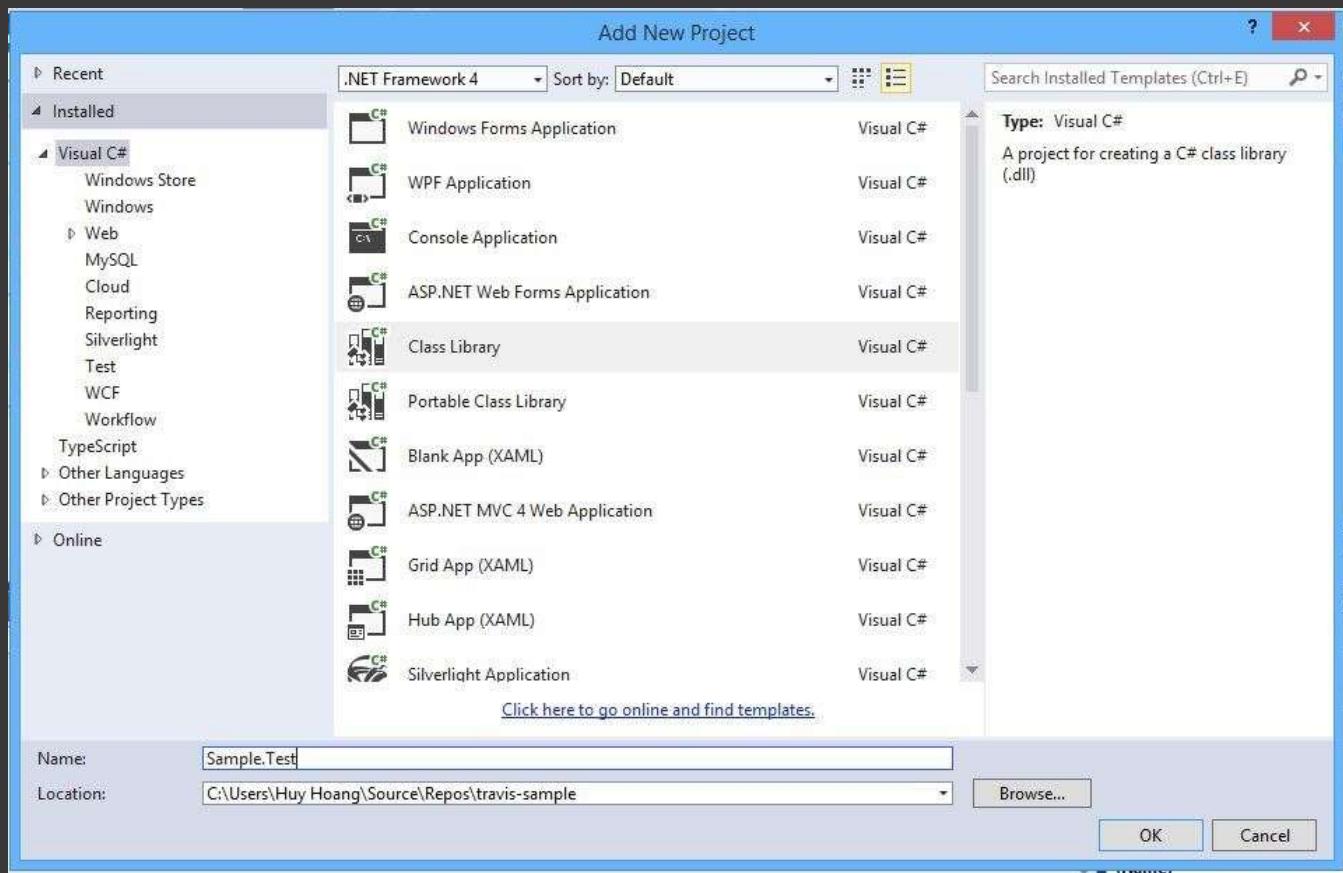
Trước đây, để viết Unit Test trong C#, ta thường phải tạo một project test riêng, sử dụng thư viện MSTest của Microsoft. MSTest hỗ trợ khá nhiều chức năng: Test dữ liệu từ database, đo performance hệ thống, xuất dữ liệu report. Tuy nhiên, do MSTest đi kèm với Visual Studio, không thể chạy riêng rẽ, lại khá nặng nề, do đó NUnit được

nhiều người ưa thích hơn. NUnit có 1 bộ runner riêng, có thể chạy UnitTest độc lập không cần VisualStudio, ngoài ra nó cũng hỗ trợ một số tính năng mà MSTest không có (parameter test, Assert Throw).

Nếu bạn chưa bao giờ viết Unit Test, bắt đầu với NUnit cũng là 1 lựa chọn không tồi. Đầu tiên, ta hãy tạo 1 project console trong Visual Studio. Ta viết 1 class Calculator với các 2 hàm Add và Subtract:

```
1  public class Calculator
2  {
3      public int Add(int x, int y)
4      {
5          return x + y;
6      }
7
8      public int Subtract(int x, int y)
9      {
10         return x - y;
11     }
12
13 }
```

Bây giờ, ta sẽ viết UnitTest để test 2 hàm này. Thông thường, code Unit Test sẽ được nằm ở một project Test riêng biệt, do đó ta sẽ tạo 1 project mới, thêm “.Test” đằng sau tên. Các bạn nhớ chọn kiểu project là Class Library nhé.



Ở project mới, bấm vào Tools -> Library Package Manager -> Package Manager Console, gõ vào: `Install-Package NUnit -Version 2.6.4` để cài đặt NUnit nhé. Ta tạo 1 class mới có tên là CalculatorTest. Bắt đầu viết thôi:

```
1  using NUnit.Framework;
2
3  [TestFixture]
4  public class CalculatorTest
5  {
6      private Calculator _cal;
7
8      [SetUp]
9      public void Setup()
10     {
11         _cal = new Calculator();
12     }
13
14     [Test]
15     public void OnePlusOneEqualTwo()
16     {
17         Assert.AreEqual(2, _cal.Add(1, 1));
18     }
19
20     [Test]
21     public void TwoPlusTwoEqualFour()
22     {
23     }
```

```
20             Assert.AreEqual(4, _cal.Add(2, 2));
21         }
22
23         [Test]
24         public void FourPlusOneEqualFive()
25         {
26             Assert.AreEqual(5, _cal.Add(4, 1));
27         }
28
29
30
31 }
```

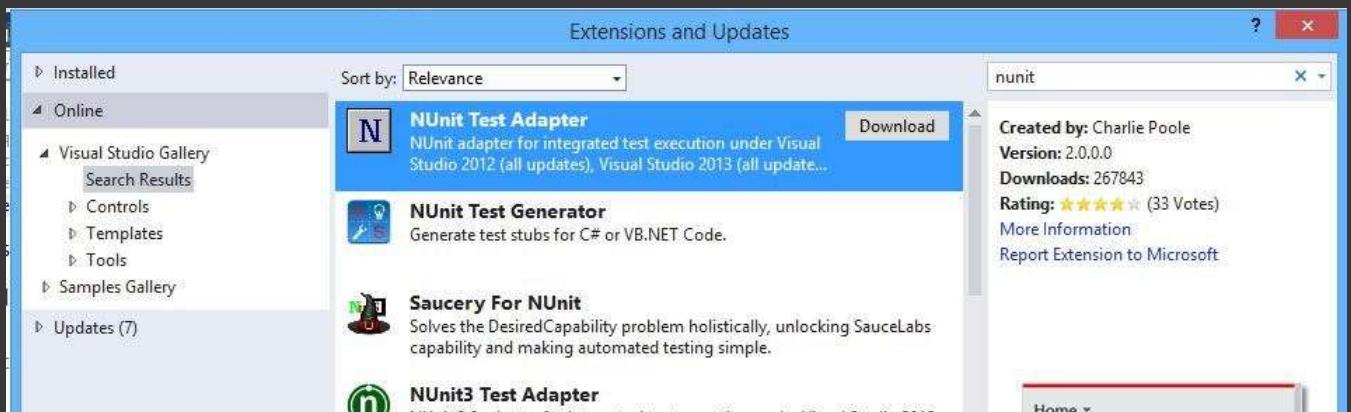
Giải thích một số annotation:

- Annotation `[TestFixture]` đặt vào đầu class chứa các unit test, đánh dấu đây là một bộ các unit test.
- Annotation `[SetUp]` để đánh dấu hàm setup. Hàm này sẽ được chạy vào đầu mỗi unit test.
- Annotation `[Test]` để đánh dấu hàm bên dưới là một unit test. Nhiều khi bạn viết unit test xong nhưng không thấy hiện unit test lên là do quên annotation này nhé.

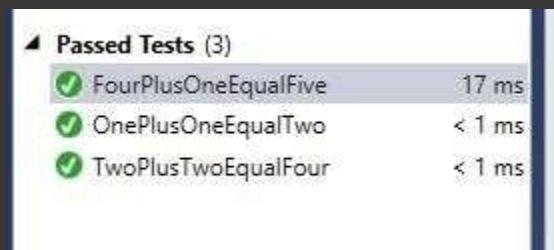
Để chạy Unit Test, ta cần cài *NUnit.TestAdapter*, đây là bộ runner cho phép chạy các Unit Test của NUnit trong Visual Studio. Bạn cũng có thể dùng 1 trong 2 cách sau:

1. Tải [NUnit GuiRunner](#) về, mở file dll của project test, các unit test sẽ hiện ra để chạy.
2. Tải Reshaper. Reshaper cũng tích hợp sẵn 1 bộ Runner cho phép chạy Unit Test của NUnit.

Để cài TestAdapter cho Visual Studio, ta vào Tool, Extentions and Updates, tìm NUnit và cài đặt. Cài xong nhớ Restart lại Visual Studio nhé.



Bấm Test -> Run -> All Tests để bắt đầu chạy. Ta sẽ thấy toàn bộ các case đã pass, màu xanh hiền hòa như trong ảnh.



Ta có ý sửa lại code để code chạy sai. Chạy lại Unit Test, ta sẽ thấy các case đã fail. Unit Test đã tự động bắt lỗi chương trình cho bạn rồi đấy. Bấm vào từng test case, ta sẽ thấy kết quả mong muốn ở đoạn “**Expected**”, kết quả code chạy ở đoạn “**But was**“.

The screenshot shows the Visual Studio IDE interface. On the left, the Test Explorer window displays three failed tests: 'FourPlusOneEqualFive' (39 ms), 'OnePlusOneEqualTwo' (1 ms), and 'TwoPlusTwoEqualFour' (1 ms). The 'Failed Tests (3)' section is highlighted with a red border. On the right, the code editor shows the 'Calculator.cs' file. The 'Add' method contains a bug where it returns $x + y * 9$ instead of $x + y$. The 'Output' window at the bottom shows the build log and test results.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace TravisSample
{
    public class Calculator
    {
        public int Add(int x, int y)
        {
            return x + y * 9; // Bug: multiplies by 9 instead of 1
        }

        public int Subtract(int x, int y)
        {
            return x - y;
        }
    }
}
```

Output:

```
1>----- Build started: Project: TravisSample, Co
1> TravisSample -> C:\Users\Huy Hoang\Source\Re
2>----- Build started: Project: TravisSample.Tes
2> Restoring NuGet packages...
2> To prevent NuGet from downloading packages du
2> All packages listed in packages.config are al
2> TravisSample.Test -> C:\Users\Huy Hoang\Souc
===== Build: 2 succeeded, 0 failed, 0 up-to-
```

Mục đích của bài viết là hướng dẫn các bạn nhập môn việc viết Unit Test cho C# bằng NUnit nên mình sẽ dùng ở đây. Nếu được ủng hộ, mình sẽ giới thiệu một số kĩ thuật phức tạp hơn như: Sử dụng Mock Object, Parameterized Test, ... ở những bài sau. Bạn nào muốn viết unit test cho javascript có thể thử tìm hiểu về [Jasmine](#), mình đã có 2 bài giới thiệu tổng quát. Cảm ơn và hẹn gặp lại.

Top 6 “trường dạy code” cho các developer

Posted on 20/08/2015 by Pham Huy Hoàng

Là một developer, việc học 1 ngôn ngữ, công nghệ mới là “chuyện thường ở huyệt”. Minh đã từng chia sẻ một số hướng tiếp cận ngôn ngữ, công nghệ ở [bài trước](#). Bài viết này sẽ giới thiệu **1 số “trường code” online**. Các trường này cung cấp bài giảng online dưới dạng video (có hoặc không có phụ đề), cho phép ta code trực tiếp trên trình duyệt. Bảng xếp hạng này dựa theo độ nổi tiếng của web trên google, cũng như trải nghiệm của mình khi sử dụng.

Các “trường code” này đều là tiếng Anh nhé, vì mình không có thói quen học **hay tìm tài liệu bằng tiếng Việt**. Không phải mình kỉ thị tiếng Việt hay đâu, vì trước giờ tiếng Việt không bao giờ cung cấp đủ tài liệu cho mình học cả. Không tin thì các bạn thử tìm tài liệu tiếng Việt đầy đủ về Ionic Framework hay Caliburn.Micro xem :(.

[Code Academy](#)

The screenshot shows the Code Academy homepage. At the top, there's a navigation bar with the logo 'codecademy' on the left, 'LOGIN' in the middle, and 'SIGN UP' on the right. Below the navigation, a section titled 'Recommended For You' features a circular icon with a laptop and the text 'Build a Professional Website'. A brief description below the icon states: 'In this course you'll get to build the Airbnb home page and learn the fundamentals of web development in the process.' A large red 'START' button is prominently displayed. Further down the page, another section titled 'Web Developer Skills' is visible with a similar circular icon and a description: 'Learn to build professional websites and applications as used by real businesses.'

Web nổi tiếng nhất trong danh sách. Giao diện của trang khá trực quan, đơn giản, **lại miễn phí**. Bạn có thể học HTML, CSS, JS, jQuery, ... ở đây. Code Academy dạy theo kiểu: Ra một số task, chúng ta thực hiện từng task (Bằng cách code), sau khi

hoàn thành các task ta sẽ chuyển tới bài sau. Theo ý kiến cá nhân, các bạn mới nhập môn HTML CSS có thể học trang này.

Khuyết điểm: Số lượng các công nghệ/ngôn ngữ mà web dạy hơi ít.

Pluralsight

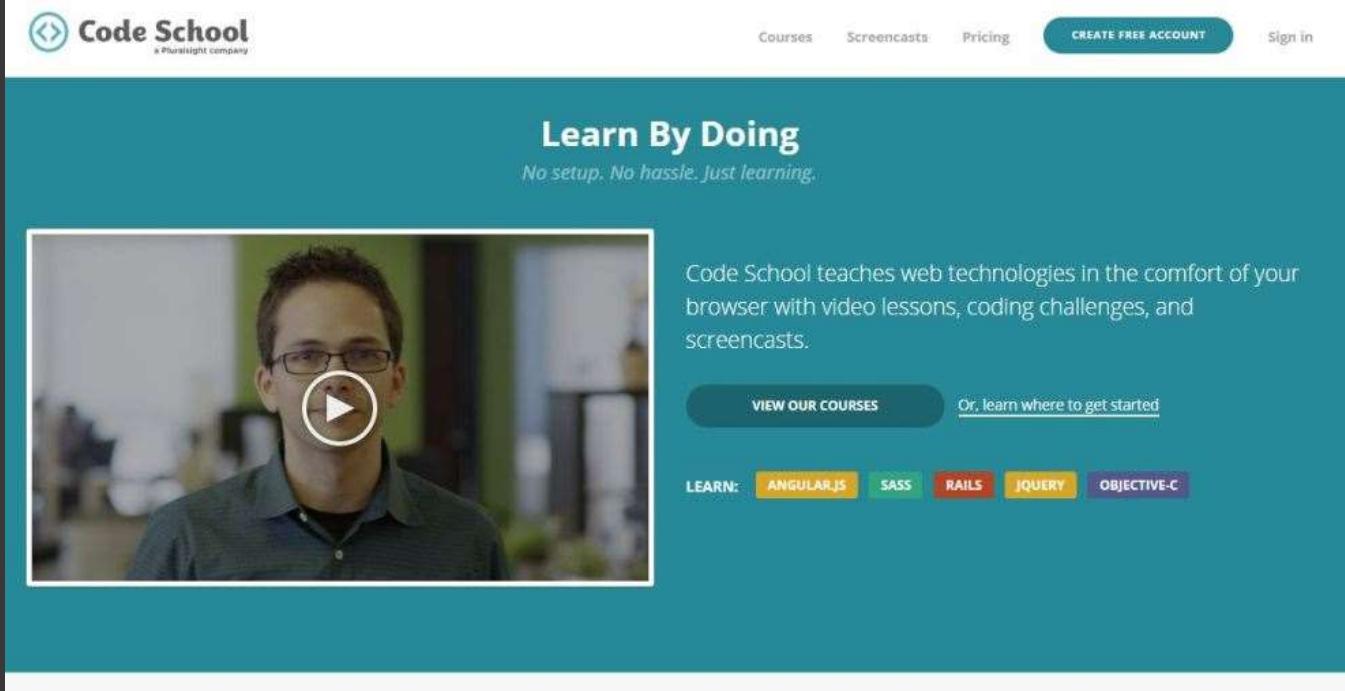


Tuy chỉ đứng thứ nhì trong bảng xếp hạng, nhưng trong cộng đồng developer, pluralsight lại nổi tiếng hơn codeacademy nhiều. Bạn có thể học được hầu như mọi ngôn ngữ, mọi framework (C#, Java, PHP, MVC, AngularJS, ...) trên pluralsight. Tác giả các khóa học đều là những lập trình viên chuyên nghiệp, danh tiếng (MVP), chất lượng khóa học cũng rất cao.

Pluralsight có thu phí đăng ký. Giá dao động khoảng 30-50USD/tháng. Các bạn sinh viên có thể dùng tài khoản trường để được free 2 tháng. Dân đi làm như mình có thể xin ké tài khoản của cty để vào. Mình học được vô số thứ hữu ích từ trang này: AngularJS, định hướng sự nghiệp, Dependency Injection,.... Bạn nào cần giới thiệu 1 số course hay cứ liên hệ mình.

Khuyết điểm: Học qua video, không thực hành. Tốn phí, phải dùng tài khoản VIP mới download được bài tập.

Code School

The screenshot shows the homepage of Code School. At the top, there's a navigation bar with links for 'Courses', 'Screencasts', 'Pricing', 'CREATE FREE ACCOUNT', and 'Sign in'. The main title 'Learn By Doing' is prominently displayed in a large, bold font, with the tagline 'No setup. No hassle. Just learning.' underneath it. To the left of the text is a video thumbnail featuring a man with glasses. Below the video thumbnail are two buttons: 'VIEW OUR COURSES' and 'Or, learn where to get started'. A horizontal menu at the bottom lists various technologies: 'LEARN:' followed by 'ANGULAR.JS', 'SASS', 'RAILS', 'JQUERY', and 'OBJECTIVE-C'.

Một website khá mới mẻ, đang gây được sự chú ý trong thời gian gần đây (Vừa được Pluralsight mua lại).

Codeschool dạy khá nhiều: Ruby, Python, AngularJS, Javascript, một số course miễn phí, nhưng phần lớn là thu phí >.<. Hình thức học cũng na ná code academy, bạn sẽ được giao một số task nhất định. Bằng cách hoàn thành từng task, chương trình sẽ chấm điểm và cho bạn qua màn mới.

Tuts+

The screenshot shows the Tuts+ website. At the top, there's a navigation bar with links for 'All Topics', 'Find tutorials, courses, and more...', 'Free Tutorials', 'Courses', 'eBooks', 'Blog', 'Pricing', 'Sign In', and 'Free Account'. Below the navigation is a large banner with the headline 'Learn Creative Skills, Shape Your Future'. A subtext below it says 'Join over 5 million people using Tuts+ each month to learn skills including code, illustration, photography, web design, and more.' There's a green button labeled 'Subscribe from \$15/month'. To the right of the text is a collage of various creative tools and concepts like books, a keyboard, a smartphone, a camera, and a clapperboard. Below the banner, there's a section titled 'Video courses to build new skills from start to finish.' with a link to 'Browse All Courses'.

Website này không chỉ dạy code mà còn có 1 số course về đồ họa, thiết kế. Mình chưa dùng bao giờ nhưng nghe nói là trang này dạy dưới dạng bài giảng hoặc ebook, thôi xin kiếu.

[Udacity](#)

The screenshot shows the Udacity website. At the top, there's a navigation bar with links for 'Nanodegree', 'Catalog', 'Sign In', and 'Sign Up'. The main visual is a man wearing VR goggles, looking at a computer screen. Overlaid text reads 'Become a Web Developer' and 'Earn a Nanodegree'. Below this is a 'Learn more' button. At the bottom of the main image, there's a white bar with logos for 'Courses built by Google', 'facebook', 'mongoDB', 'cloudera', and 'at&t'.

Trang web này khá nổi tiếng trong cộng đồng lập trình viên nước ngoài. Số lượng course không nhiều, nhưng được tập trung thành các pathway, hỗ trợ kiến thức cho nhau. Udacity còn có hệ thống nanodegree, tương đương với những chứng chỉ-mini

dành cho các lập trình viên. Có điều cái giá hơi cao, **tận 200USD/tháng**, do đó mình không khuyến khích các bạn học trang này.

Skillshare



Mình chưa xài trang này bao giờ, nhưng đọc qua review thì thấy nó cũng khá tốt. Trang web dạy thông qua các bài giảng dạng video, có bài tập và giáo viên chấm điểm. Giá khá rẻ so với Udacity phía trên, chỉ 10\$/tháng.

Còn các bạn thì sao? Hãy chia sẻ những “trường code” mà các bạn hay học trong phần comment của bài viết nhé.

Những kỹ năng cần có của một web developer

Posted on 18/08/2015 by Pham Huy Hoàng

Hiện nay, một lập trình viên có thể lựa chọn cho mình nhiều hướng phát triển: Lập trình nhúng (Embeded System), lập trình web, lập trình ứng dụng di động, ... Vì mình đi theo hướng lập trình web, mình sẽ chia sẻ một số kỹ năng mà các bạn cần chuẩn bị nếu muốn theo con đường web developer.



Kỹ năng front-end

Nói đơn giản: Front-end là những gì người dùng nhìn thấy và tương tác. Nó là “mặt tiền” của một trang web. Nếu bạn thích thiết kế, gần gũi với người dùng thì bạn có thể tập trung phát triển những kỹ năng front-end, trở thành một front-end developer (Lương cũng cao lắm đấy nhé). Những kỹ năng bạn cần phát triển bao gồm:

- HTML/CSS/Javascript cơ bản (Đừng nghĩ js dễ nhẹ, khó lắm đấy).
- Một số thư viện/framework nổi tiếng: Bootstrap, jQuery, AngularJS, EmberJS.
- Kỹ năng thiết kế, sử dụng Photoshop. Kiến thức và kinh nghiệm về UI/UX.
- LESS, SASS (stylesheet language).
- Sử dụng npm, grunt, ... để optimize, minimize HTML/CSS/JS.
- Kiến thức về Ajax, cách thiết kế giao diện responsive

Vai trò của front-end trong 1 dự án là khá quan trọng, vì giao diện là thứ đập vào mắt người dùng đầu tiên. Front-end developer không chỉ thiết kế giao diện đẹp, mà còn

phải rõ ràng, dễ sử dụng. Người dùng có thể làm việc mình muốn một cách đơn giản, nhanh gọn (Google là một ví dụ).

Một số sách hay để nâng cao kỹ năng front-end:

- Series Head First, The Missing Manual (Head First HTML & CSS, jQuery The Missing Manual ...)
- Don't make me think
- The Design of Everyday Things



Kỹ năng back-end

Back-end là những thứ người dùng không nhìn thấy, nhưng giúp cho hệ thống hoạt động trơn tru. Dữ liệu của người dùng, thuật toán phân tích ... đều nằm ở back-end. Nếu front-end là lớp son, lớp vỏ của một ngôi nhà thì back-end chính là giàn giáo, xương sườn của ngôi nhà đó. Những kỹ năng bạn cần có gồm có:

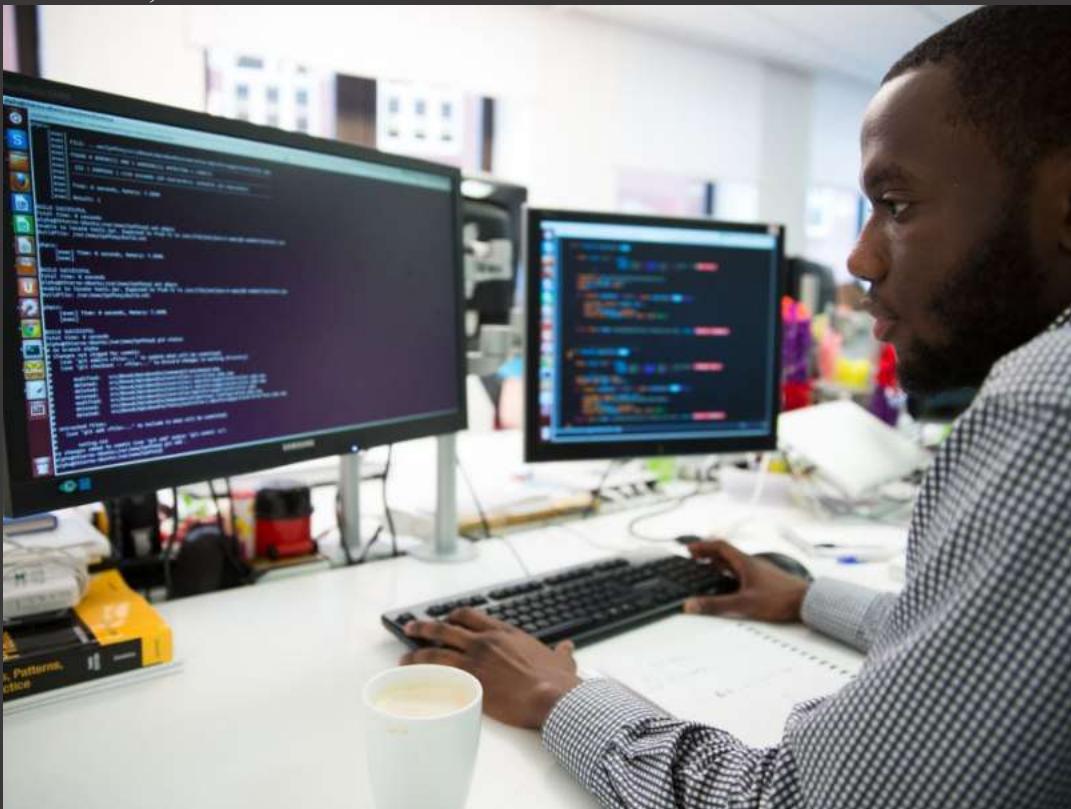
- Ngôn ngữ server-side để viết back-end: C#, Java, Python, Ruby, Dĩ nhiên là phải bao gồm kiến thức về những web framework đi kèm các ngôn ngữ này: ASP.NET MVC, Spring, Django, Rails ...
- Kiến thức về database SQL: MS SQL Server, MySQL, ... Gần đây một số database NoSQL đang khá thịnh hành: Neo4j, MongoDB, ...

- Kiến thức về web nói chung, cách viết Web Service, cách đăng nhập và phân quyền .
- Kiến thức về 1 số CMS: WordPress, Joomla, Umbraco,

Kiến thức phần back-end rất nhiều và phức tạp, do đó một back-end developer chỉ nên tập trung vào 2-3 ngôn ngữ chính, đừng ráng ôm hết kéo “táu hỏa nhập ma”. Code phần back-end thường rất nhiều và “khủng”, do đó cần có cấu trúc tốt, dễ cải tiến và mở rộng (bằng cách áp dụng SOLID). Back-end developer có thể trau dồi kiến thức để leo lên vị trí System Analyst hoặc Software Architecture.

Một số sách hay cho back-end developer:

- Clean Code
- Code Complete
- Head First Design Pattern
- Sách chuyên sâu về ngôn ngữ/framework: C# in Depth, Pro .NET 4.5, Spring in Action, ...



Kỹ năng phân tích thiết kế

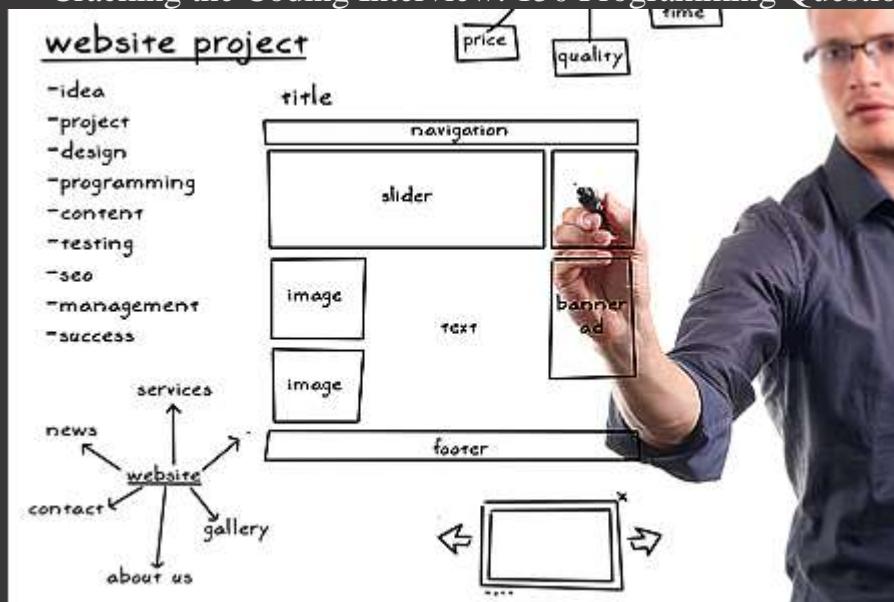
Hiện nay, ranh giới giữa front-end và back-end trong lập trình web khá mong manh. Đa phần các web developer thường giỏi về back-end, có khá kiến thức về front-

end, việc này khá hữu dụng. Biết cả front-end và back-end, bạn sẽ biết được một trang web hoạt động như thế nào – từ đầu tới cuối.

Lập trình viên front-end, back-end cũng có thể “lấn sân” qua mảng mobile nhờ sự giúp sức của một số framework như Cordova (HTML, CSS, JS), Ionic, Window Phone App (C#), ... Để tăng giá trị của bản thân, ngoài kỹ năng cứng, bạn cần trau dồi kỹ năng phân tích, giải quyết vấn đề : Khách hàng cần gì ở trang web, lượng truy cập là bao nhiêu, làm sao để tăng performance. Nhà tuyển dụng sẽ đánh giá kỹ năng này của bạn khi phỏng vấn đấy.

Một số sách nên tham khảo:

- The Pragmatic Programmer: From Journeyman to Master
- The Passionate Programmer: Creating a Remarkable Career in Software Development
- Getting Real
- Cracking the Coding Interview: 150 Programming Questions and Solutions



Bài viết này được viết theo yêu cầu của bạn Phước Lê đã comment trong bài "Kỷ niệm post thứ 50". Rất mong nhận được sự góp ý và ủng hộ từ các bạn.

Sự khác biệt giữa Web Site và Web Application

Posted on 13/08/2015 by Phạm Huy Hoàng

Hiện nay một số bạn học ngành IT vẫn còn lầm lẫn giữa khái niệm website và web app, sẵn tiện có một bạn hỏi nên mình viết bài này nhân tiện giải thích luôn.

Đây là một câu hỏi “tưởng dễ mà không phải dễ”, bởi vì ranh giới giữa website và webapp khá mong manh. Mình phải tổng hợp khá nhiều câu trả lời từ [stackoverflow](#) và [programmers.stackexchange](#) mới đưa ra được một câu trả lời “gần đúng” nhất.

1. Khái niệm website

Ngày xưa ngày xưa, khi Internet còn thô sơ, web được viết bằng html đơn lẻ. Mỗi trang web đơn lẻ được viết bằng html gọi là **Web Page**. Tập hợp nhiều trang web đơn lẻ, thành một trang web lớn, có chung tên miền, được gọi là **Website**. VD đơn giản: Mỗi bài viết trên blog của mình chính là một web page, tập hợp toàn bộ các bài viết lại chính là một website, tên là toidicodedao.com.

How Much Does a Website Cost???



2. Khái niệm webapp

Đầu tiên, ta hãy xem lại khái niệm application (trên wiki).

Ứng dụng là một loại chương trình có khả năng làm cho máy tính thực hiện trực tiếp một công việc nào đó người dùng muốn thực hiện

Ban đầu, các website chỉ bao gồm text, hình ảnh và video, liên kết với nhau thông qua các link. Tác dụng của website là lưu trữ và hiển thị thông tin. Người dùng chỉ có thể đọc, xem, click các link để di chuyển giữa các page.

Về sau, với sự ra đời của các ngôn ngữ server: CGI, Perl, PHP, ... các website đã trở nên “động” hơn, có thể tương tác với người dùng. Từ đây, người dùng có thể dùng web để “*thực hiện một công việc nào đó bằng máy tính*”, do đó web app ra đời.



Nói dễ hiểu, web app là những ứng dụng chạy trên web. Thông qua web app, người dùng có thể thực hiện một số công việc: tính toán, chia sẻ hình ảnh, mua sắm ... **Tính tương tác của web app cao hơn website rất nhiều.**

Với một số người không rành về IT, tất cả những thứ online, vào được bằng trình duyệt đều là website cả. Do đó họ thường yêu cầu bạn là: website quản lý siêu thị, website bán hàng, ... thực chất chúng đều là webapp hết.

3. So sánh website và web app

Trên thực tế, ranh giới giữa web app và website khá mong manh. Một trang báo mạng – vnexpress chẳng hạn, trong mắt người đọc nó là website. Nhưng trong mắt biên tập viên hoặc admin, nó lại là web app. Một số trang web cho phép người dùng search, comment nhưng nó vẫn chỉ là website, chưa phải là webapp. Dưới đây là bảng so sánh (tương đối).

Web site

Tính tương tác thấp, ít chức năng (Xem, đọc, click qua lại giữa các link...)

Web app

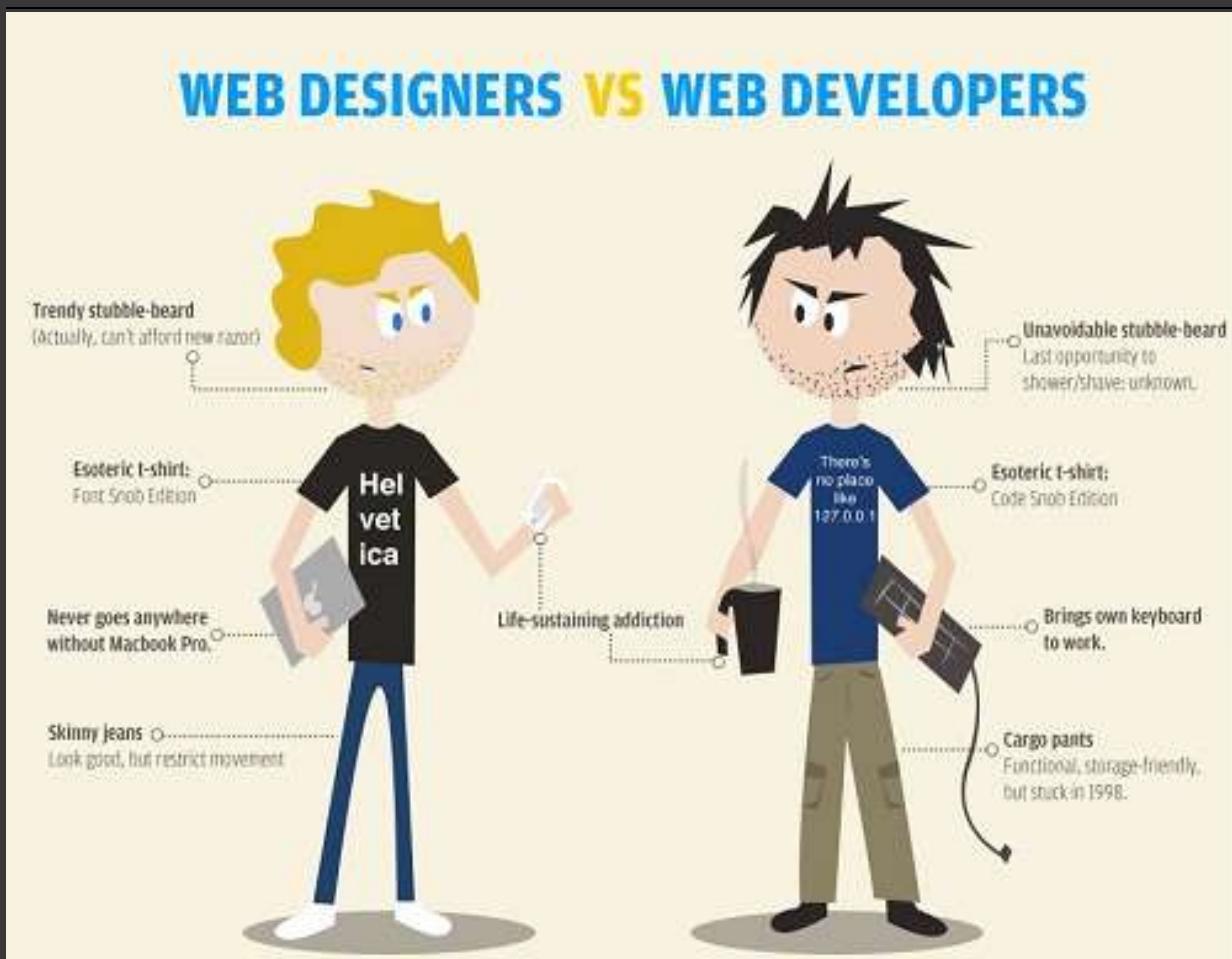
Tính tương tác cao, nhiều chức năng (Đăng thông tin, upload file, xuất báo cáo...)

Được tạo thành từ các trang html tinh và một số tài nguyên (hình ảnh, âm thanh, video)

Được tạo bởi html và code ở back end (PHP, C#, Java, ...)

Được dùng để lưu trữ, hiển thị thông tin

Được dùng để “thực hiện một công việc”, thực hiện các chức năng của một ứng dụng



Ý sau của câu hỏi: ”Những kỹ năng cần có nếu muốn trở thành Web Developers” sẽ được trả lời trong bài viết sau nhé. Bài viết này được viết theo yêu cầu của bạn Phước Lê đã comment trong bài “Kỷ niệm post thứ 50” nhé.

Tạo động lực học tập và làm việc – Sức mạnh của thói quen

Posted on 06/08/2015 by Phạm Huy Hoàng

Thời sinh viên, đã bao giờ bạn muốn làm bài tập, ôn thi ngay nhưng lại bị games, đi chơi, gái gú cảm dỗ chưa?

Thời sinh viên, đã bao giờ bạn muốn lấy một tấm bằng ngoại ngữ, một chứng chỉ, nhưng tìm học được nửa tiếng rồi lại thôi chưa?

Lúc đi làm, đã bao giờ bạn muốn học một công nghệ mới, một ngôn ngữ mới, nhưng được một vài hôm lại thấy chán nản và muốn bỏ chua?

Nếu câu trả lời của bạn là “Có”, đừng lo, chẳng có gì xấu hổ cả, ngày xưa mình cũng từng như bạn (giờ vẫn vậy). Tuy nhiên, nhờ một vài bí quyết đơn giản, mình đã cảm thấy tự tin, dễ dàng khi học và tiếp thu kiến thức mới, có được một công việc kha khá, cũng nhu một số tấm bằng kha khá.

TOEIC

**LISTENING AND READING
INSTITUTIONAL PROGRAM SCORE REPORT**

Pham Huy Hoang

Name: Pham Huy Hoang	LISTENING		TOTAL SCORE	
Identification Number: 241238503	Date of Birth: 1992/03/04	Your score: 475	Score: 5	945
Identification Number: 2013/08/13	Date of Birth: 2013/08/13	READING	Your score: 470	Score: 5
Test Date: Unspecified	Test Date: Unspecified		Score: 495	

LISTENING

This section scores a maximum of 450 points. Test takers who score around 450 typically have the following range of listening skills: accurate, basic, limited, or very limited. They can understand some words and short phrases in a variety of situations, but they cannot understand most words and cannot understand longer stretches of speech. They can often do this even when the difference in level is not extreme. They can understand some words and short phrases in a variety of situations, but they cannot understand most words and cannot understand longer stretches of speech. They can understand some words and short phrases in a variety of situations, but they cannot understand most words and cannot understand longer stretches of speech. They can understand some words and short phrases in a variety of situations, but they cannot understand most words and cannot understand longer stretches of speech. They can understand some words and short phrases in a variety of situations, but they cannot understand most words and cannot understand longer stretches of speech.

READING

This section scores a maximum of 450 points. Test takers who score around 450 typically have the following range of reading skills: accurate, basic, limited, or very limited. They can read simple texts in one or two sentences, but they can make little inference. They can understand factual information, even when it is presented in a variety of situations. They can understand some words and short phrases in a variety of situations, but they cannot understand most words and cannot understand longer stretches of speech. They can also understand some words and short phrases in a variety of situations, but they cannot understand most words and cannot understand longer stretches of speech. They can also understand some words and short phrases in a variety of situations, but they cannot understand most words and cannot understand longer stretches of speech. They can also understand some words and short phrases in a variety of situations, but they cannot understand most words and cannot understand longer stretches of speech.

ABILITIES MEASURED

PERCENT CORRECT OF ABILITIES MEASURED	
Can relate past, present, and future events on information that is related by another in a short sentence	84
Can understand simple, concrete, and familiar situations on information that is explicitly stated or implied	81
Can understand clearly in short, familiar contexts	30
Can understand clearly in short, unfamiliar contexts	57
Can understand clearly in extended, unfamiliar contexts	57

ABILITIES MEASURED

PERCENT CORRECT OF ABILITIES MEASURED	
Can relate information based on imagination or written text	87
Can understand abstract concepts, ideas, and situations, in written texts	52
Can understand abstract concepts, ideas, and situations, in a dialogue between two or more people	84
Can understand abstract concepts, ideas, and situations, in written texts	33
Can understand grammatical structures in written texts	91

* Proficiency Description Table can be found on our web site, www.ets.org/toeic.

HOW TO READ YOUR SCORE REPORT:

Percent Correct of Abilities Measured: Percentage of items you answered correctly. At this test form for each line of the table, the first number is the percent correct of items you answered correctly. Below these abilities cannot be attributed to the performance of test takers at this level, but may be used to compare your performance to other test takers.

Note: TOEIC scores below 100 points are not counted for reporting purposes.

INTERNATIONAL ENGLISH LANGUAGE TESTING SYSTEM

Test Report Form

ACADEMIC

NOTE: Admission to undergraduate and post graduate courses should be based on the ACADEMIC Reading and Writing Modules. GENERAL TRAINING Reading and Writing Modules are not designed to test the full range of language skills required for academic purposes. It is recommended that the candidate's language ability as indicated in the Test Report Form be re-assessed after two years from the date of the test.

Centre Number: VN101 Date: 28/FEB/2015 Candidate Number: 010060

Candidate Details

Family Name: PHAM First Name: HUY HOANG Candidate ID: 241238503



Date of Birth: 04/03/1992 Sex (M/F): M Scheme Code: Private Candidate

Country or Region of Origin:

Country of Nationality: VIET NAM

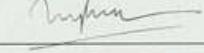
First Language: VIETNAMESE

Test Results

Listening: 8.0 Reading: 9.0 Writing: 6.5 Speaking: 6.0 Overall Band Score: 7.5

Administrator Comments

Administrator's Signature: 

Date: 12/03/2015 Test Report Form Number: 14VN010060PHAH101A



The validity of this IELTS Test Report Form can be verified online by recognising organisations at <http://ielts.ucles.org.uk>

Đăng hình đê các bạn khỏi bảo là mình ngồi không chém gió nữa (Tự học nên các bạn đừng nhò mình tư vấn trung tâm nhé). Bí quyết tạo động lực học của mình không quá phức tạp.

Hãy đặt ra những mục tiêu ngắn hạn, dài hạn cho mình. Hãy nghĩ tới niềm vui mình sẽ có khi đạt được mục tiêu, những buồn bức thất vọng khi không hoàn thành. Sau đó lên kế hoạch và thực hiện mục tiêu đó.

Ban đang gật gù: “Ô, ra là vậy, từ giờ mình sẽ chăm chỉ đặt mục tiêu, chăm chỉ làm theo”. Xin lỗi, **đoạn trên mình chém gió đấy, cái thứ “mục tiêu” sáo rỗng đó chẳng giúp ích cho bạn mấy đâu**. Đặt mục tiêu được 1,2 ngày bạn sẽ lại chán và bỏ cuộc cho xem.



Vô số sách vở đã nói về tầm quan trọng “mục tiêu”, cho rằng nó là kim chỉ nam thành công. Cá nhân mình lại cho rằng chúng ta đã quá đề cao nó. Đặt ra mục tiêu sẽ đặt áp lực lên bản thân bạn. Khi không đạt được mục tiêu đó, bạn sẽ dễ thấy thất vọng về bản thân. Đặt ra mục tiêu, ta nghĩ rằng mình có thể dự đoán được tương lai, trong khi “thế sự vô thường, đường đời khó đoán”. Cách mà mình tạo tự động lực, ép bản thân tiến bộ, đó là:

Thay vì đặt mục tiêu, dồn động lực để thực hiện, hãy xây dựng các thói quen. Chỉ cần gắng sức xây dựng được thói quen tốt, bạn sẽ dần dần đi đúng hướng.

Nếu để đam mê dẫn đường, bạn sẽ cháy hết sức mình vào một công việc nào đó. Rồi sau 1 tuần, 1 tháng, khi đam mê tắt dần, bạn sẽ thấy nản và chán, rồi bỏ dở. Ngược lại, để thói quen dẫn đường, mình luôn bắt đầu bằng mọi việc những bước tiến nhỏ, từ từ.

good habits
ARE AS
ADDICTIVE
as bad habits
BUT MUCH MORE REWARDING.

Bản thân mình định lập blog từ tháng 10 năm ngoái, nhưng ngần ngừ lười viết tới tháng 12 mới làm. Sau khi bắt đầu viết, mình chỉ cố gắng duy trì thói quen ra 2 bài/tuần. Đến bây giờ, mình đã hình thành được thói quen ngồi viết blog mỗi thứ 3 thứ 5. Dù nhiều khi mình **không có động lực**, cũng chẳng mấy hào hứng khi ngồi vào máy, mình vẫn ngồi xuống và viết, vì nó đã thành một thói quen.

Đơn giản vậy đấy. Mình dễ học công nghệ mới vì luyện được thói quen đọc sách công nghệ, xem video pluralsight. Mình tự học được tiếng Anh vì luyện được thói quen bỏ học AV 2 tiếng mỗi ngày sau khi đi làm về. Khi đã thành thói quen, bạn có thể **học tập, làm việc dễ dàng, không cần động lực**. Điều mình thật sự muốn chia sẻ là *Hãy để đam mê và động lực chỉ đường cho bạn thấy điều bạn muốn làm, chứ đừng để chúng dẫn đường. Chính kỉ luật và thói quen mới là người dẫn đường cho bạn*.

MOTIVATION

IS WHAT GETS YOU STARTED,



HABIT

IS WHAT KEEPS YOU GOING

Dần dần, bạn sẽ xây dựng lòng tin vào bản thân – *tôi có thể biến mọi chuyện thành thói quen, tôi có thể làm mọi thứ*. Tin mình đi, mọi thứ chỉ mệt mỏi vào giai đoạn khởi đầu thôi. Khi tất cả đã thành một thói quen, bạn có thể làm những chuyện “cực nhọc, khó khăn” một cách vui vẻ và thích thú.

Tái bút: Mình cũng đang muôn tập thói quen dậy sớm mỗi sáng chạy bộ 2 cây số mà đang lười, chắc phải cố gắng thêm. Dù biết là chỉ cần ráng 1 tuần, mọi thứ sẽ thành thói quen nhưng mình vẫn khoái ngủ nướng hơn. Có thể do mình hơi lười vận động chăng?



Bài viết tham khảo từ các nguồn sau đây. Các bạn nên đọc bản gốc, có nhiều ý khá hay và đáng học hỏi:

<http://jamesclear.com/why-is-it-so-hard-to-form-good-habits>

<http://jamesclear.com/goals-systems>

<http://zenhabits.net/no-goal/>

Một cuốn sách khá hay về việc xây dựng thói quen là [Superhuman by Habit: A Guide to Becoming the Best Possible Version of Yourself, One Tiny Habit at a Time](#), các bạn nên tìm đọc. Bài viết này được viết theo yêu cầu của bạn flyl9134 đã comment trong bài “[Kỷ niệm post thứ 50](#)” nhé.

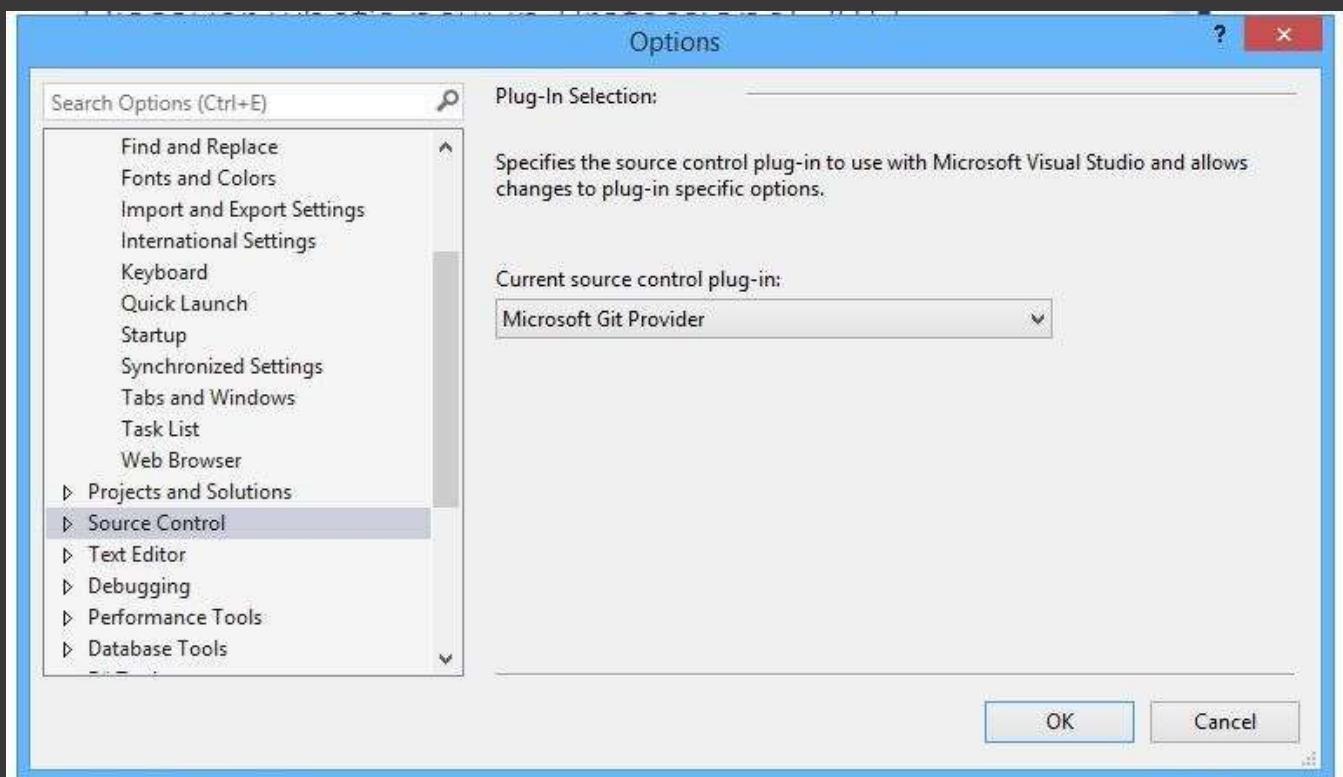
[Tutorial] Hướng dẫn tích hợp Visual Studio với Github

Posted on 04/08/2015 by Phạm Huy Hoàng

Trước đây, để quản lý source code, ta thường sử dụng SVN, host toàn bộ source code trên google code. Trong vòng nhiều năm gần đây, Git đang trở thành 1 xu thế mới, thay thế dần cho SVN. Hầu như các thư viện javascript, css nổi tiếng hiện giờ đều đặt đại bản doanh trên github. Google Code sẽ đóng cửa vào năm sau, vì vậy hầu như các project mới bây giờ đều được host trên Github. Mình viết bài này nhằm hướng dẫn các bạn dùng Visual Studio có thể lấy code, submit code lên github dễ dàng với Visual Studio nhé.

Phiên bản VS hỗ trợ Github là bản VS 2012, 2013 và bản 2015 mới nhất.

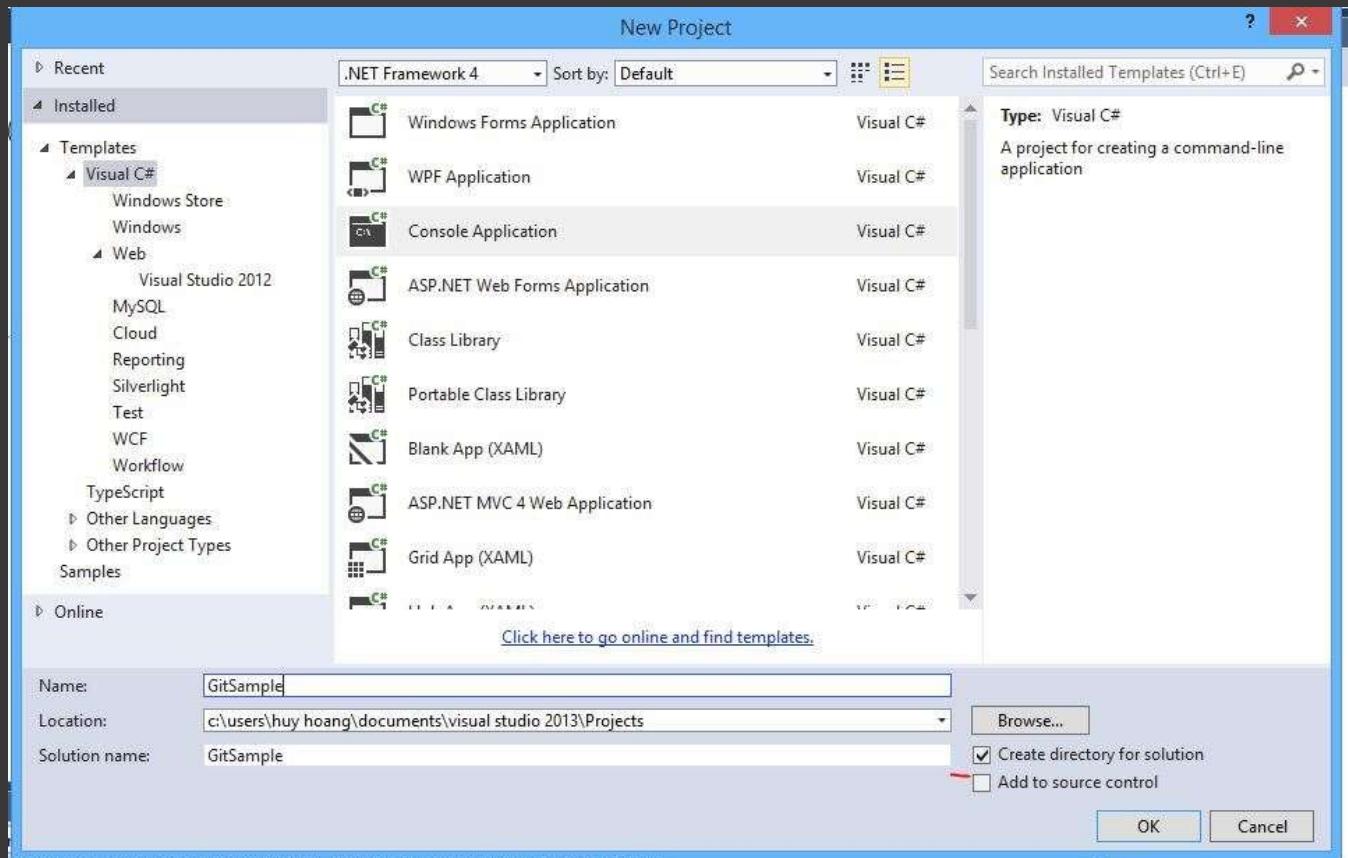
Đầu tiên, bạn cần vào Tool -> Options -> Source Version Control, chọn Microsoft Git Provider.



Bài viết sẽ hướng dẫn các bước cơ bản để làm việc với git:

1. Tạo 1 project mới trên github.
2. Lấy code của 1 project trên github về máy.
3. Sửa code, update và commit (Pull và push trong Git).

Bước 1: Tạo 1 project C# mới, ở đây mình tạo Console cho nhẹ, nhớ bỏ dòng Add to Source Control nhé.



Sau khi tạo 1 project mới, mình sẽ add thêm thư viện Newtonsoft.Json. Các bạn hãy dùng Nuget để tải thư viện. Nếu ta add thư viện dưới dạng file, khi commit code lên git sẽ khá nặng. Khi dùng Nuget, mọi thư viện tải về sẽ được nuget quản lý, ta không cần commit các file thư viện lên. Khi người khác tải về và build, Nuget sẽ tự tải thư viện cần thiết.

```

Package Manager Console
Package source: nuget.org
Default project: GitSample

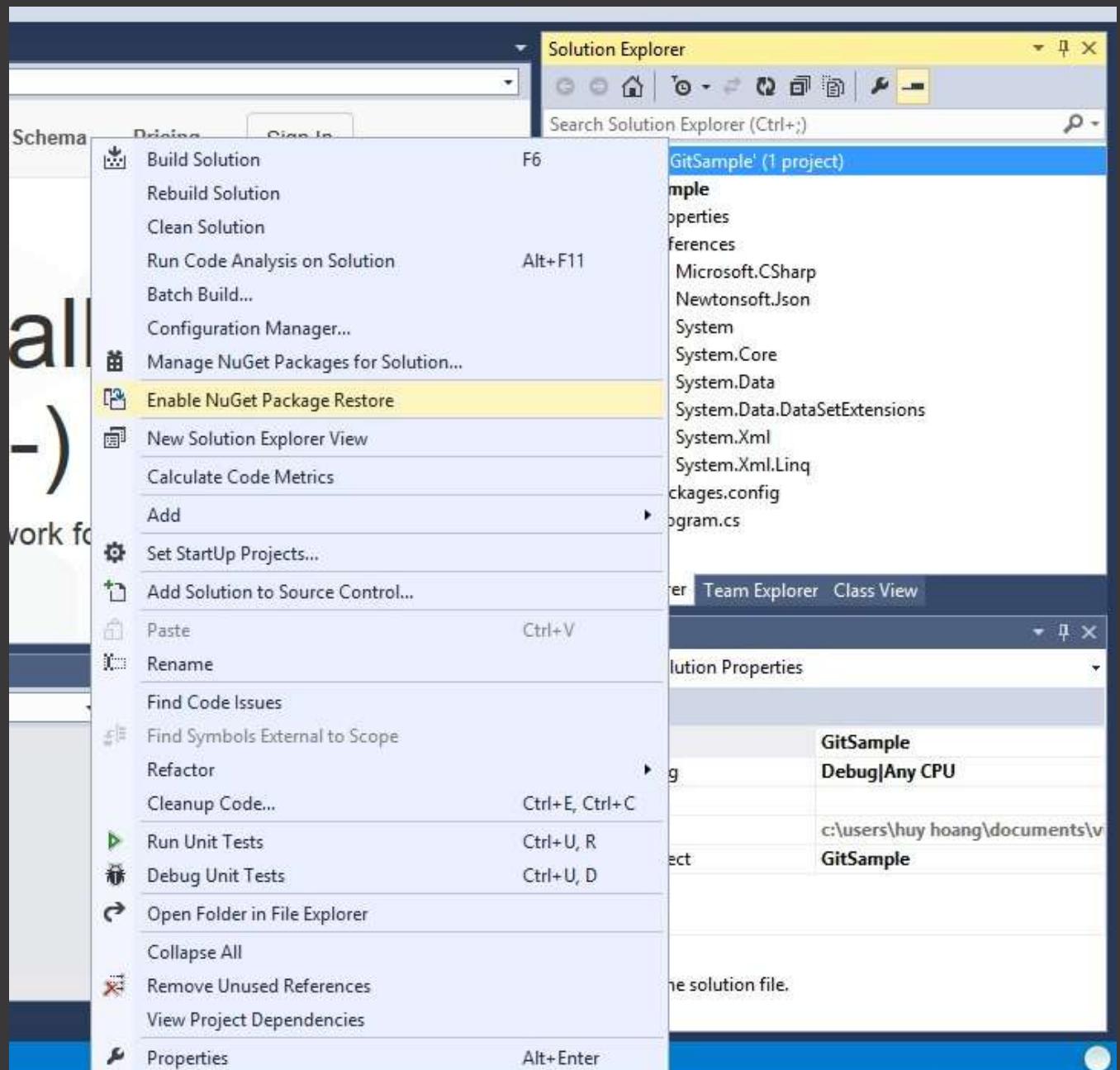
Package Manager Console Host Version 2.7.40911.287
Type 'get-help NuGet' to see all available NuGet commands.

PM> Install-Package Newtonsoft.Json
Installing 'Newtonsoft.Json 7.0.1'.
Successfully installed 'Newtonsoft.Json 7.0.1'.
Adding 'Newtonsoft.Json 7.0.1' to GitSample.
Successfully added 'Newtonsoft.Json 7.0.1' to GitSample.

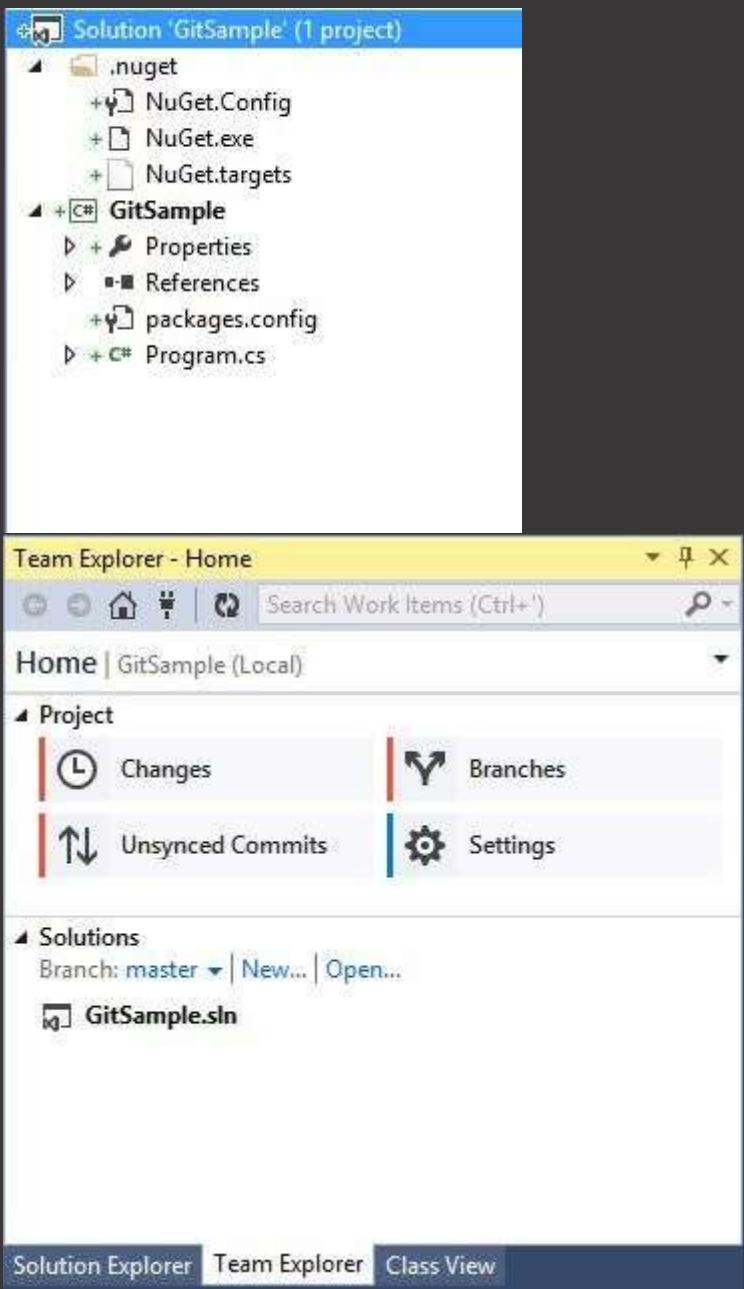
100 %

```

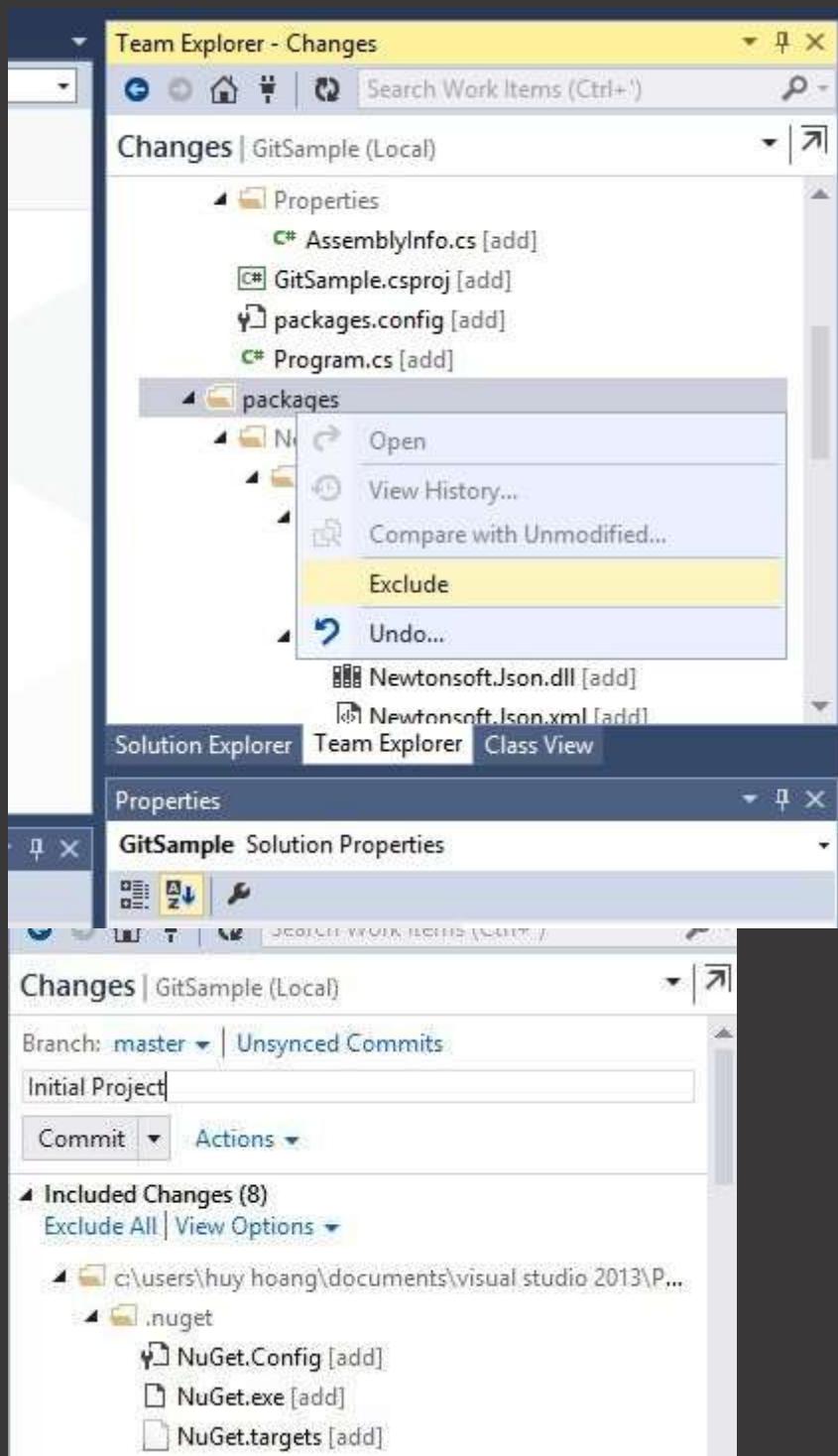
Nhớ bấm chuột phải vào Solution, chọn “Enable Nuget package restore” và chọn Yes nhé:

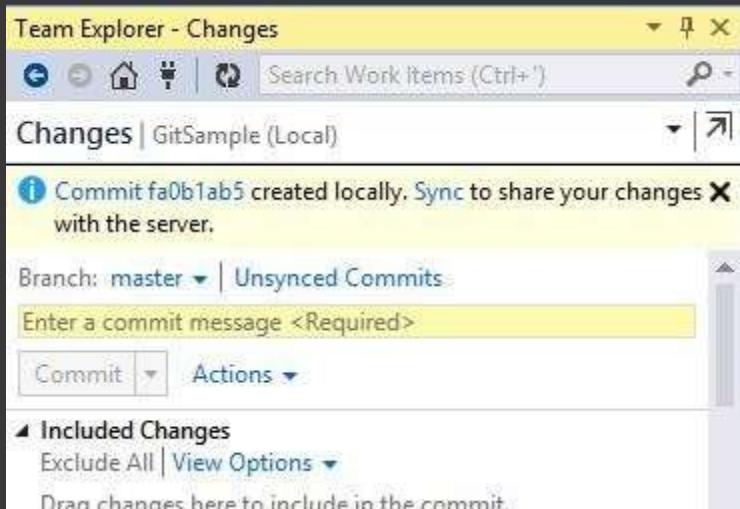


Project giờ đã hoàn thành, ta bấm chuột phải vào solution, chọn “Add Solution to Source Control”, chọn “Git” và bấm OK. Ta sẽ thấy dấu + hiện lên bên trái các file, đánh dấu đây là file mới. Ta bấm qua Tab “Team Explorer”, bấm Change để xem danh sách các file.



Chuột phải vào thư mục packages, bấm Exclude để loại bỏ thư mục đó ra, sau đó điền comment và commit code nhé.





Ta đã commit code thành công, nhưng đây chỉ là local repository, tức là repository trên máy bạn mà thôi.

Tiếp theo, chúng ta sẽ tạo 1 repository trên github để commit code lên. Ta lêngithub.com, tạo 1 git mới, nhớ **tuyệt đối không đánh vào ô “Initialize readme.md”**, làm vậy bạn sẽ không commit được đâu. Bạn sẽ thấy project mới được tạo.

The image shows two screenshots of the GitHub interface. The top screenshot is a 'Create repository' form where a user is creating a new repository named 'GitSample'. The bottom screenshot is the main page for the repository 'GitSample' under the owner 'ToDiCodeDaoSampleCode'.

Create Repository Form (Top Screenshot):

- Owner:** ToDiCodeDaoSampleCode
- Repository name:** GitSample (checkmark)
- Description (optional):** (empty text area)
- Visibility:** Public (radio button selected) - Anyone can see this repository. You choose who can commit.
- Visibility:** Private (radio button) - You choose who can see and commit to this repository.
- Initialize this repository with a README:** (checkbox checked) - This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.
- Add .gitignore:** None
- Add a license:** None

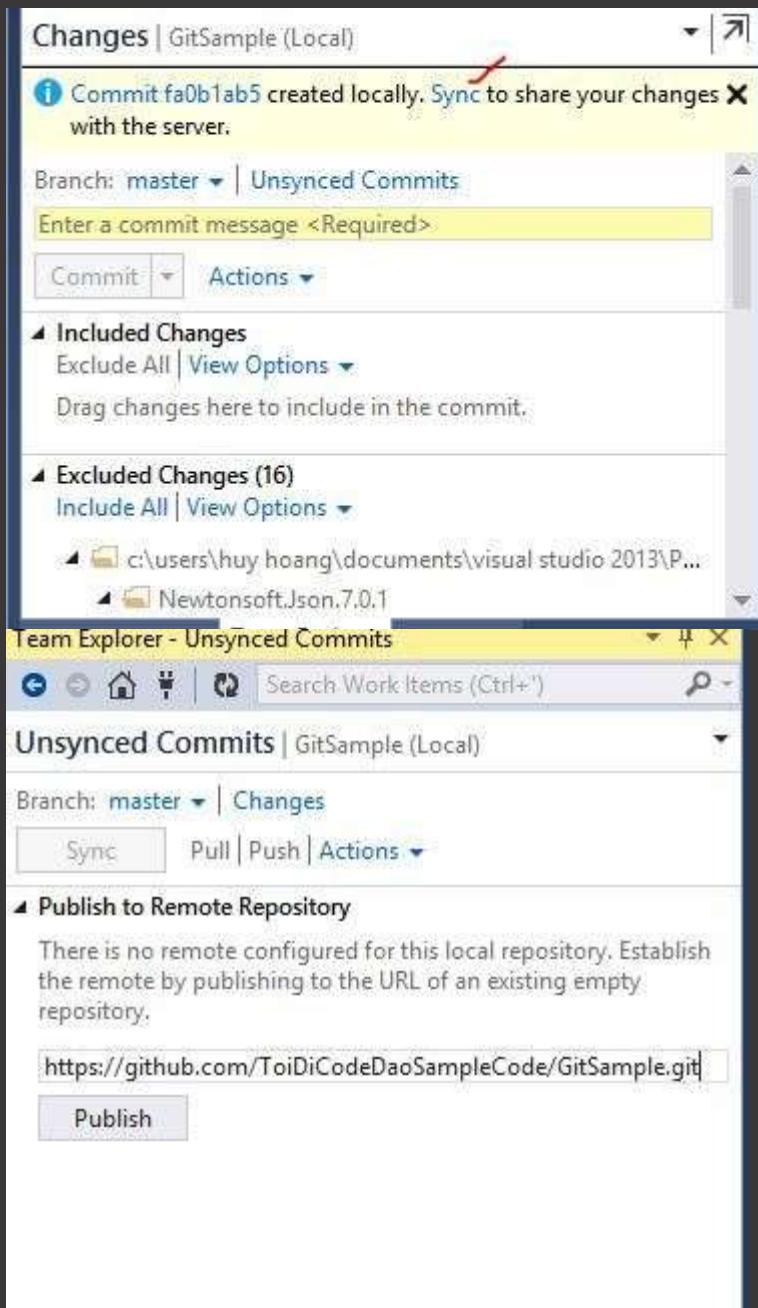
Create repository button (green)

Repository Details Page (Bottom Screenshot):

- Repository Name:** GitSample
- Owner:** ToDiCodeDaoSampleCode
- Quick setup — if you've done this kind of thing before:**
 - Set up in Desktop (button)
 - or
 - HTTPS (selected)
 - SSH
 - https://github.com/ToDiCodeDaoSampleCode/GitSample.git
- We recommend every repository include a README, LICENSE, and .gitignore.
- ...or create a new repository on the command line:**

```
echo # GitSample >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/ToDiCodeDaoSampleCode/GitSample.git
git push -u origin master
```
- ...or push an existing repository from the command line:** (button)
- Repository Sidebar:**
 - Code
 - Issues (0)
 - Pull requests (0)
 - Wiki
 - Pulse
 - Graphs
 - Settings

Hãy copy cái đóng <https://github.com…>; .Đó là đường link dẫn tới repository github của bạn. Quay lại Visual Studio, bấm nút “Sync” màu xanh xanh ấy. Dán đường dẫn github bạn đã copy vào, bấm Publish, code sẽ được đồng bộ từ local repository lên repository trên github.



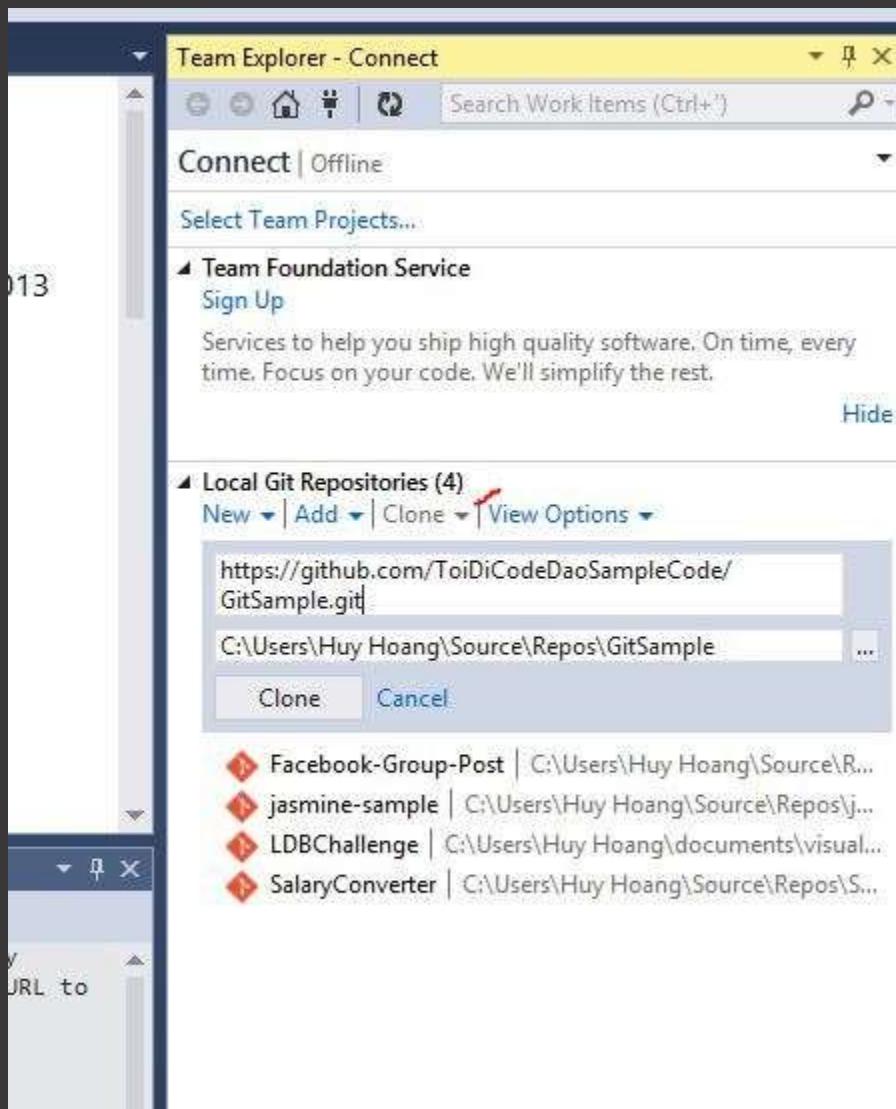
Kết quả:

The screenshot shows a GitHub repository named 'GitSample'. A yellow banner at the top indicates 'Unsynced Commits | GitSample (Local)'. It says 'The origin remote has been added and the current branch has been published.' Below this, the 'master' branch is selected. There are two commits listed under 'Initial Project': one for '.nuget' and one for 'GitSample'. Both were authored by 'conanak99' 7 minutes ago. On the right side, there are links for 'Issues', 'Pull requests', 'Wiki', 'Pulse', 'Graphs', and 'Settings'. At the bottom, there's a 'HTTPS clone URL' field with the value 'https://github.com/' followed by a copy icon. Below it are buttons for 'Clone in Desktop' and 'Download ZIP'.

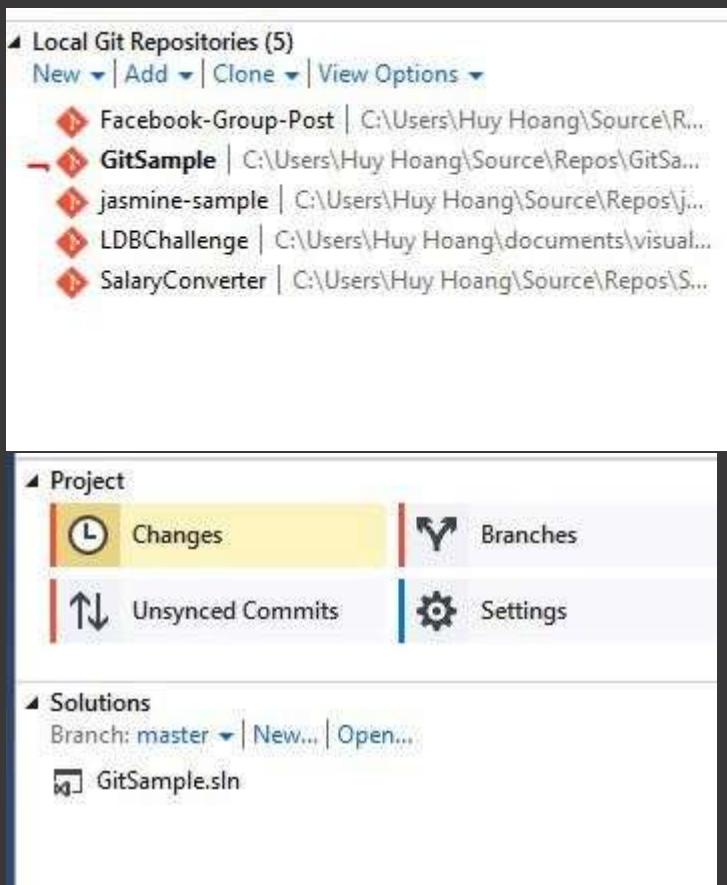
Bước 2: Cách lấy code từ github về local.

Sau khi đã đưa code lên github, bạn copy link HTTPS clone URL (góc dưới bên phải), gửi cho bạn bè.

Các bạn bật Visual Studio lên, mở lại tab Team Explorer, bấm vào hình các phích cắm (Connect to team projects). Chọn clone, sau đó dán url vào:



Sau khi đã lấy code về, nhấp vào project GitExample, sau đó nhấp vào fileGitSample.sln để mở project.



Tiến hành build, ta thấy Nuget sẽ tự động tải packages Newtonsoft.Json về.

```
Show output from: Build
1>----- Build started: Project: GitSample, Configuration: Debug Any CPU -----
1> Restoring NuGet packages...
1> To prevent NuGet from downloading packages during build, open the Visual Studio Options dialog, click on the Packag...
1> Installing 'Newtonsoft.Json 7.0.1'.
1> Successfully installed 'Newtonsoft.Json 7.0.1'.
1> GitSample -> C:\Users\Huy Hoang\Source\Repos\GitSample\GitSample\bin\Debug\GitSample.exe
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

Bước 3: Chính sửa code, commit lại code trên github.

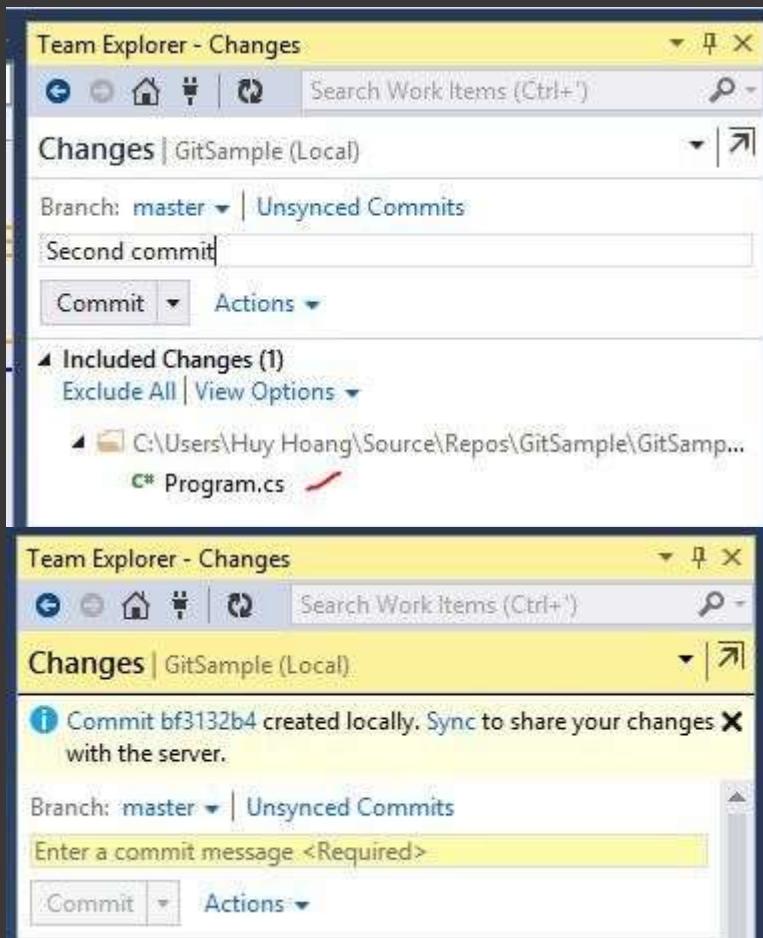
Quy trình làm việc với github như sau:

1. Chính sửa code trên máy với Visual Studio.
2. Commit code đã sửa vào local repository.
3. Đồng bộ giữa local repository và github repository.

Mình sẽ chỉnh 1 chút code trong file Program.cs. File có màu đỏ đánh dấu đã chỉnh rồi.

```
namespace GitSample
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
        }
    }
}
```

Bấm qua Tab “Team Explorer”, bấm vào Change. Nhập comment vào rồi Commit.
Commit rất nhanh (Chưa tới 1s), vì ta chỉ commit trên local.



Bấm Sync để đồng bộ dữ liệu lên github. Visual Studio sẽ hỏi username và password github của bạn. Nhớ kêu bạn bè add vào project thì mới commit code được nhé. Kết quả cuối cùng:



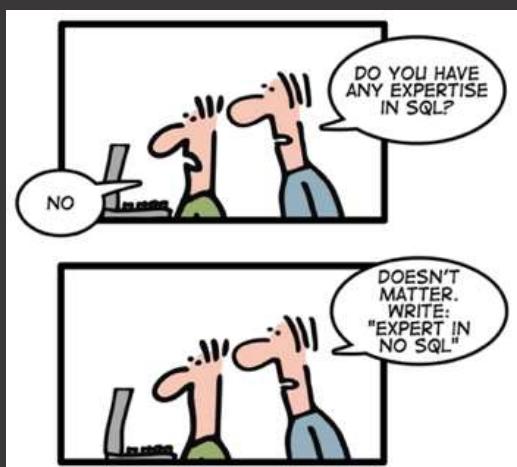
Vì Visual Studio hướng tới sự đơn giản, dễ sử dụng nên nó **không có command line**. Nếu muốn sử dụng Git Command Line, các bạn hãy tải thêm plug-in cho VS. Bài viết này là kết thúc, mong rằng nó sẽ có ích cho công việc của các bạn.

NoSQL có gì hay ho – Tổng quan về NoSQL – Phần 1

Posted on 24/09/2015 by Phạm Huy Hoàng

NoSQL đang dần nổi lên như một thế lực trong giới lập trình. Nhà nhà quảng cáo NoSQL, người người sử dụng NoSQL. MEAN stack (MongoDB, Express, AngularJS, NodeJS) đang dần lấn lướt, thay thế cho LAMP stack (Linux, Apache, MySQL, PHP/Python) đã lỗi thời.

Bài viết này giới thiệu tổng quan về NoSQL, giúp các bạn có thêm kiến thức khi phỏng vấn xin việc hoặc chém gió với nhau lúc trà dư tửu hậu.



Trước NoSQL, từng có một thứ gọi là ... SQL

Nhắc lại một chút, database (**DB**) là một cơ sở dữ liệu, gồm các bảng, hàng, cột. Những thứ mình sẽ nhắc đến trong bài như: MySQL, Microsoft SQL server, MongoDB, .. là hệ quản trị cơ sở dữ liệu (**DBMS**). Các bạn đừng nhầm lẫn 2 khái niệm này.

Thật ra, SQL không phải là ... DB, cũng không phải là DBMS. Nó là viết tắt của Structure Query Language (Ngôn ngữ truy vấn câu trúc). Ngôn ngữ này truy vấn trên nền một **RDBMS** (Hệ quản trị CSDL quan hệ). Đây là thứ các bạn sinh viên được dạy trong môn “Cơ sở dữ liệu”.

Trong RDBMS, dữ liệu được lưu vào nhiều bảng. Mỗi bảng sẽ có nhiều cột, nhiều row. Ta sử dụng SQL để truy vấn như sau:

RDBMS (MySQL, Microsoft SQL Server, Oracle, ...) *được sử dụng rất rộng rãi*, trong hầu hết các ứng dụng, vì một số lý do sau:

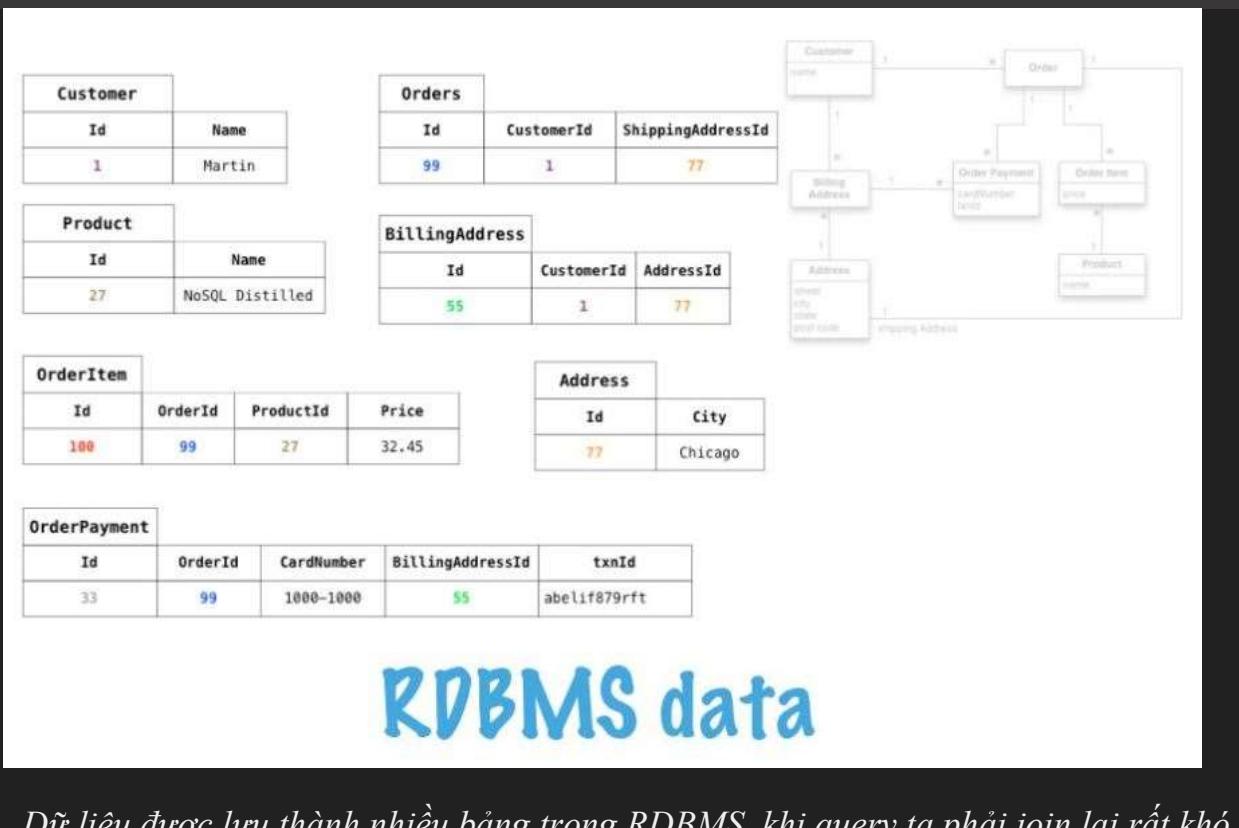
- Tính ACID (Atomicity, Consistency, Isolation, Durability) của một transaction được đảm bảo.
- Với database chuẩn 3, dữ liệu được đảm bảo tính đồng nhất và toàn vẹn (consistency).
- Có rất nhiều driver cho mọi ngôn ngữ: Java, C#, PHP.
- Số lượng lập trình viên biết và dùng SQL rất nhiềuuuuuuuuuuu.



Sau SQL, có một thứ gọi là ... NoSQL

Tuy nhiên, RDBMS vẫn còn một số khuyết điểm:

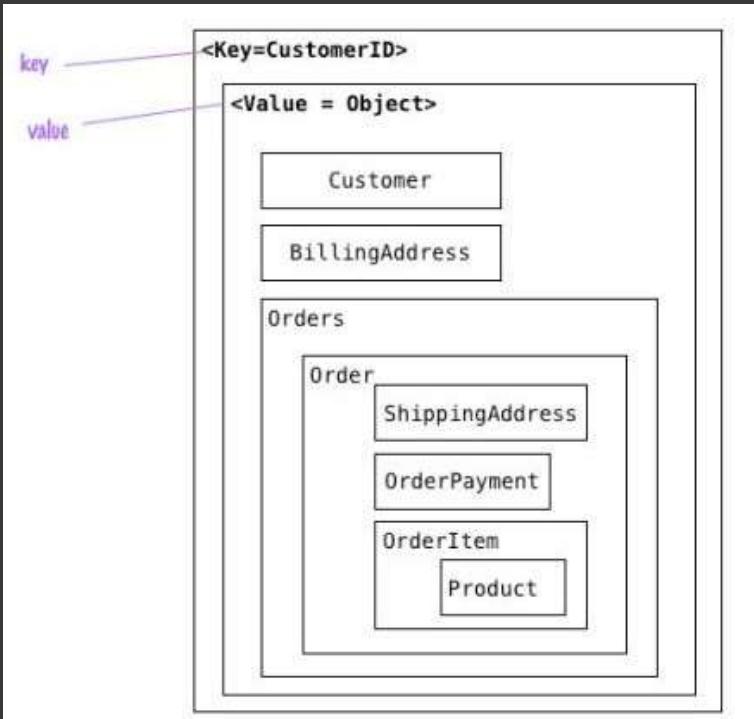
- Việc mapping giữa các bảng trong database với các object trong code khá rắc rối và phức tạp. (*Mặc dù 1 số ORM như Entity Framework, Hibernate đã đơn giản hóa chuyện này*).
- Performance sẽ bị chậm khi phải join nhiều bảng để lấy dữ liệu (Đó là lý do ta sử dụng “giảm chuẩn” để tăng hiệu suất cho RDBMS).
- Việc thay đổi cấu trúc dữ liệu (Thêm/xóa bảng hoặc thêm/xóa một field) rất mệt mỏi, kéo theo vô số thay đổi trên code.
- Không làm việc được với dữ liệu không có cấu trúc (un-structure).
- RDBMS được thiết kế để chạy trên một máy chủ. Khi muốn mở rộng, nó khó chạy trên nhiều máy (clustering).



Dữ liệu được lưu thành nhiều bảng trong RDBMS, khi query ta phải join lại rất khó khăn

NoSQL Database (Phải là **NoSQL Database** nhé) ra đời, giải quyết được những khuyết điểm của RDBMS:

- Dữ liệu trong NoSQL DB được lưu dưới dạng document, object. Truy vấn dễ dàng và nhanh hơn RDBMS nhiều.
- NoSQL có thể làm việc hoàn toàn ok với dữ liệu dạng không có cấu trúc.
- Việc đổi cấu trúc dữ liệu (Thêm, xóa trường hoặc bảng) rất dễ dàng và nhanh gọn trong NoSQL.
- Vì không đặt nặng tính ACID của transactions và tính nhất quán của dữ liệu, NoSQL DB có thể mở rộng, chạy trên nhiều máy một cách dễ dàng.



```

{
  "customer": {
    "id": 1,
    "name": "Martin",
    "billingAddress": [{"city": "Chicago"}],
    "orders": [
      {
        "id":99,
        "orderItems":[
          {
            "productId":27,
            "price": 32.45,
            "productName": "NoSQL Distilled"
          }
        ],
        "shippingAddress": [{"city":"Chicago"}]
        "orderPayment": [
          {
            "ccinfo":"1000-1000-1000-1000",
            "txnid":"abelif879rft",
            "billingAddress": {"city": "Chicago"}
          }
        ]
      }
    ]
  }
}

```

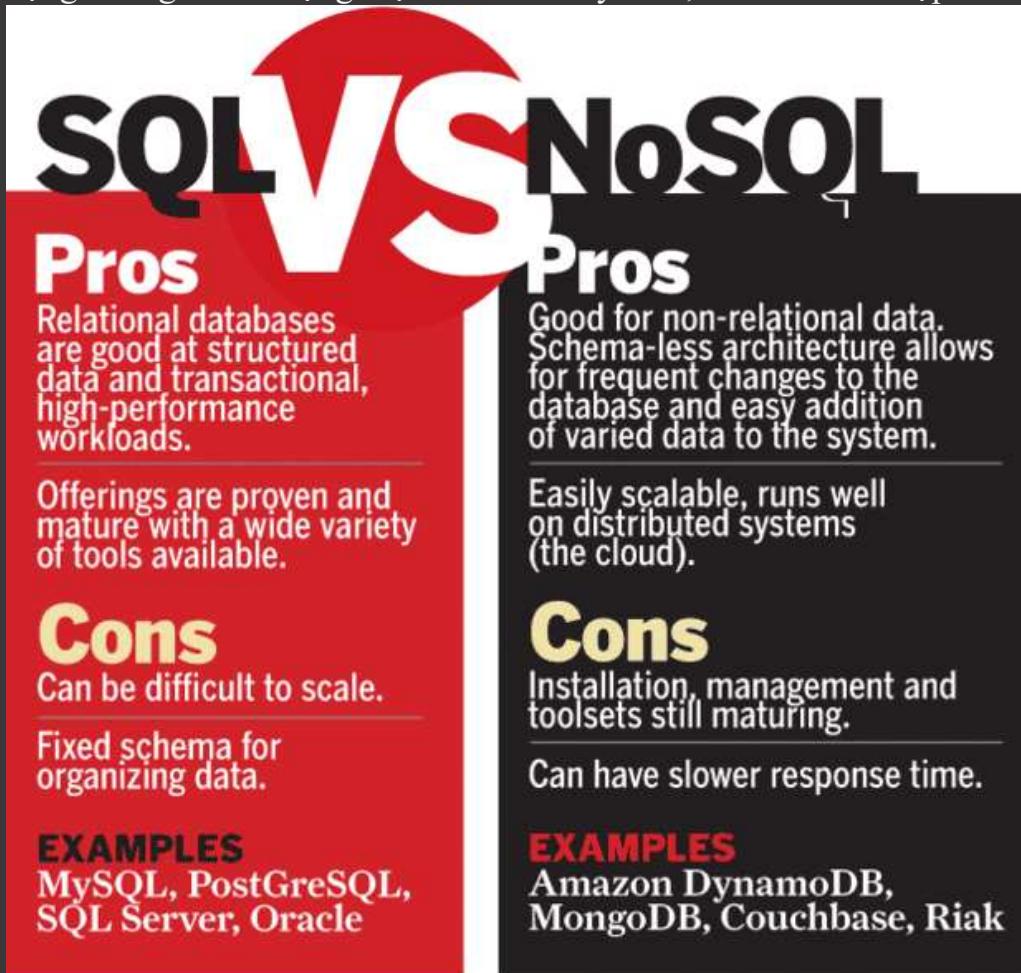
Dữ liệu được lưu dưới dạng object. Mặc dù bị trùng nhưng truy vấn rất nhanh và đơn giản.

Trong tương lai, NoSQL sẽ ... thế chỗ SQL ???

Tóm lại, NoSQL database chỉ là một kiểu database có cách lưu trữ, truy vấn dữ liệu hoàn toàn khác so với RDBMS và SQL. NoSQL bỏ qua tính toàn vẹn của dữ liệu và transaction để đổi lấy **hiệu suất nhanh và khả năng mở rộng** (scalability). Với

những ưu điểm trên, NoSQL đang được sử dụng nhiều trong các dự án Big Data, các dự án Real-time, số lượng dữ liệu nhiều.

Liệu NoSQL có thay thế được hoàn toàn RDBMS và SQL được không? **Câu trả lời là KHÔNG**. Trong tương lai, RDBMS vẫn sẽ giữ được chỗ đứng của mình. Một ứng dụng không chỉ sử dụng một database duy nhất, và có thể kết hợp cả SQL lẫn NoSQL.



Ở [bài viết sau](#), mình sẽ giới thiệu một số kiểu NoSQL Database cơ bản, cũng như các DBMS nổi tiếng và được sử dụng nhiều. Mong các bạn đón xem.

NoSQL có gì hay ho – Tổng quan về NoSQL – Phần 2

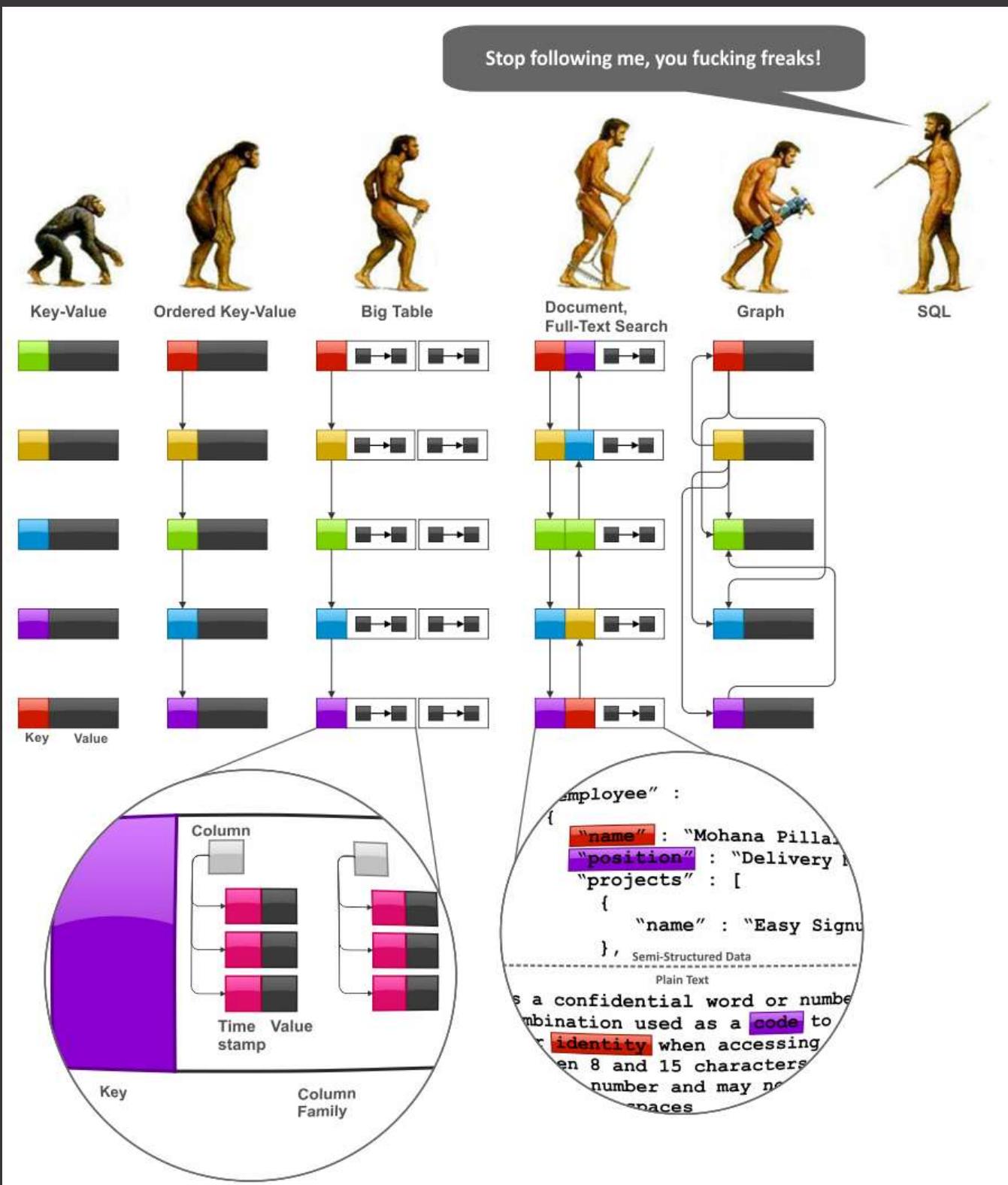
Posted on [29/09/2015](#) by [Phạm Huy Hoàng](#)

Như đã giới thiệu ở [bài trước](#), NoSQL Database đang được sử dụng ngày một nhiều hơn, chiếm dần vị trí của Relational Database. Bài viết này sẽ giới thiệu một số dạng NoSQL và ứng dụng của chúng.

Hiện nay, trên thị trường có rất nhiều NoSQL Database Management System: MongoDB, RavenDB, Redis, Neo4j,... Ta có thể chia NoSQL thành 4 loại:

- Key-Value Database
- Document Database
- Column-Family Database
- Graph Database

Stop following me, you fucking freaks!

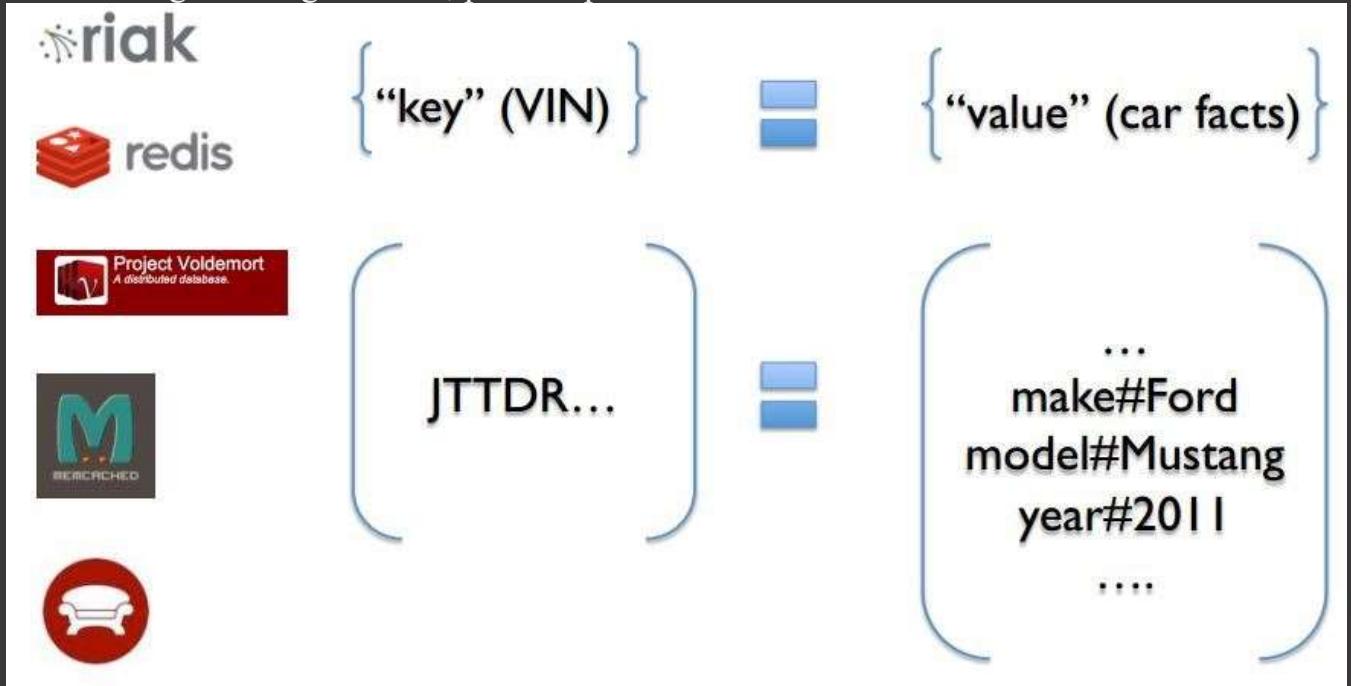


Key-Value Database

Giới thiệu: Dữ liệu được lưu trữ trong database dưới dạng key-value, giống như một [Dictionary trong C#](#). Để truy vấn dữ liệu trong database, ta dựa vào key để lấy value ra. Các database dạng này có tốc độ truy vấn rất nhanh.

Database tiêu biểu: Riak, Redis, MemCache, Project Voldemort, CouchBase

Ứng dụng: Do tốc độ truy xuất nhanh, key-value database thường được dùng để **làm cache cho ứng dụng** (Tiêu biểu là Redis và MemCache). Ngoài ra, nó còn được dùng để lưu thông tin trong sessions, profiles/preferences của user...



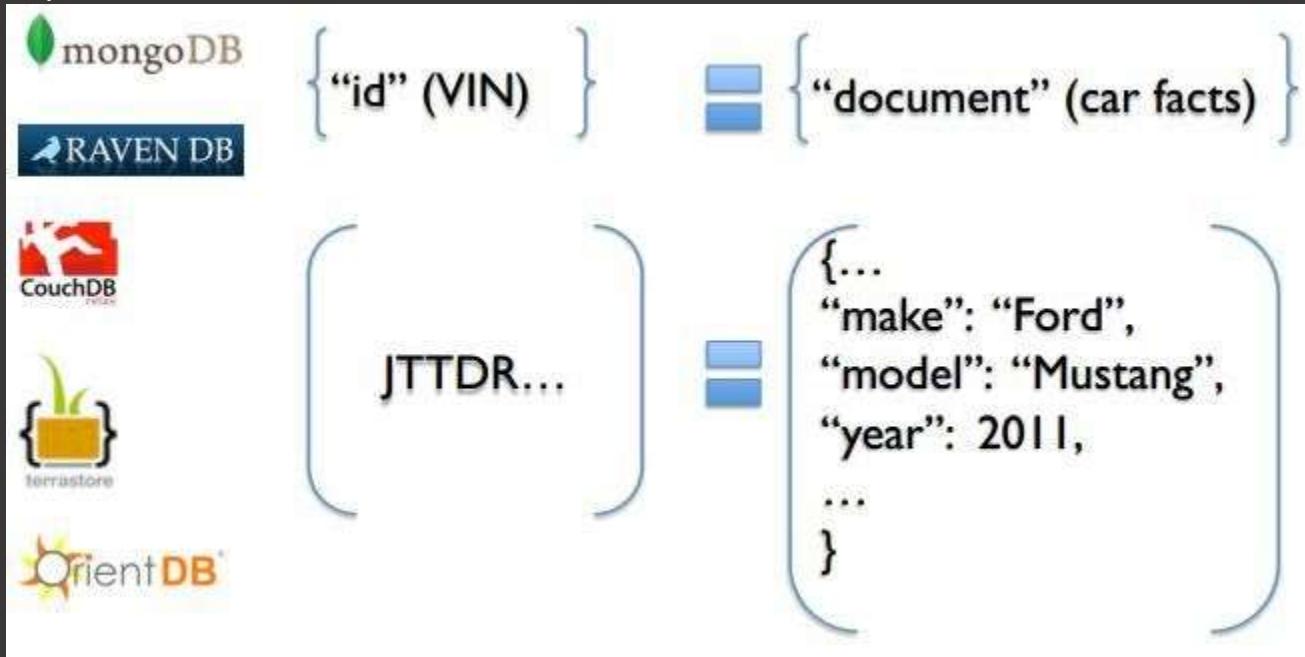
Document Database

Giới thiệu: Mỗi object sẽ được lưu trữ trong database dưới dạng một document. Dữ liệu sẽ được lưu trữ dưới dạng BSON/JSON/XML dưới database. Dữ liệu không schema cứng như SQL, do đó ta **có thể thêm/sửa field, thay đổi table**, ... rất nhanh và đơn giản. Database dạng này có tốc độ truy vấn nhanh, có thể thực hiện các câu truy vấn phức tạp, dễ mở rộng (scalability). Mỗi database có **một kiểu truy vấn riêng**, khá là loạn xà ngầu (RavenDB dùng Lucene, MongoDB lại dùng query document).

Database tiêu biểu: MongoDB, RavenDB, CouchDB, TerraStone, OrientDB

Ứng dụng: Do nhanh và linh động, document database thường đóng vai trò làm **database cho các ứng dụng prototype, big data, e-commerce, CMS**. Ngoài ra, ta

còn dùng nó để lưu log hoặc history. Mình khuyên các bạn nên thử học MongoDB hoặc RavenDB nhé.

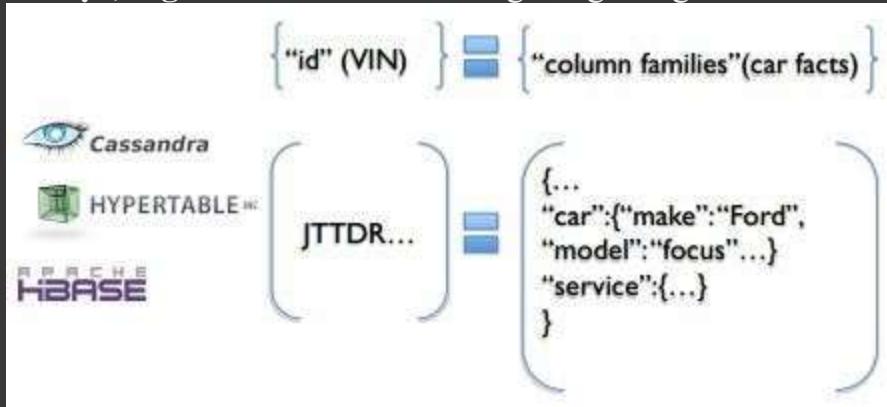


Column-Family Database

Giới thiệu: Dữ liệu được lưu trong database dưới dạng các cột, thay vì các hàng như SQL. Mỗi hàng sẽ có một key/id riêng. Điểm đặc biệt là **các hàng trong một bảng sẽ có số lượng cột khác nhau**. Câu lệnh truy vấn của nó khá giống SQL.

Database tiêu biểu: Cassandra (Phát triển bởi Facebook), HyperTable, Apache HBase

Ứng dụng: Column-Family Database được sử dụng khi ta cần **ghi một số lượng lớn dữ liệu, big data**. Nó còn được ứng dụng trong 1 số CMS và ứng dụng e-commerce.



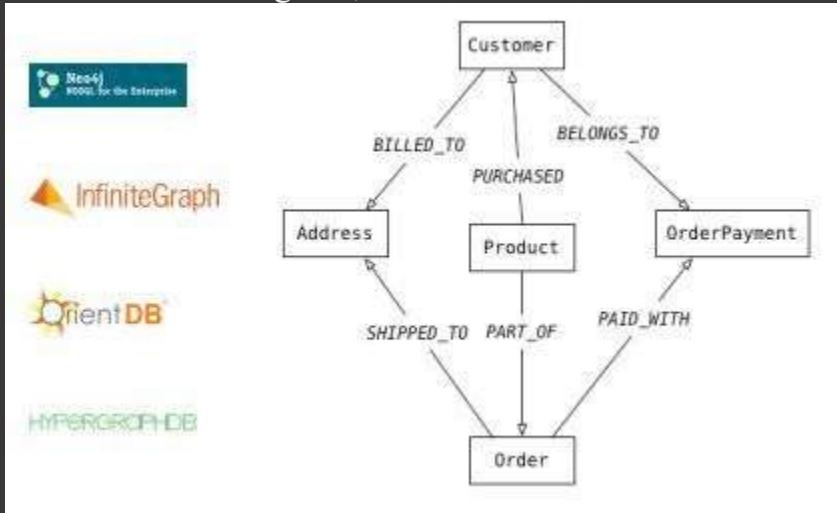
Graph Database

Giới thiệu: Dữ liệu trong graph database được lưu dưới dạng các node. Mỗi node sẽ có 1 label, 1 số properties như một row trong SQL. Các node này được kết nối với

nhau bằng các relationship. Graph database tập trung nhiều vào relationship giữa các node, áp dụng nhiều thuật toán duyệt node để tăng tốc độ.

Database tiêu biểu: Neo4j, InfiniteGraph, OrientDB, HYPERGRAPHDB

Ứng dụng: Khi cần truy vấn các mối quan hệ, graph database **truy vấn nhanh và dễ hơn nhiều** so với database. Nó được dùng trong các hệ thống: mạng nơ ron, chuyển tiền bạc, mạng xã hội (tìm bạn bè), **giới thiệu sản phẩm** (dựa theo sở thích/lịch sử mua sắm của người dùng)... Neo4j là một database free, lại có một cộng đồng rất lớn, với vô số bài hướng dẫn, các bạn nên học thử.



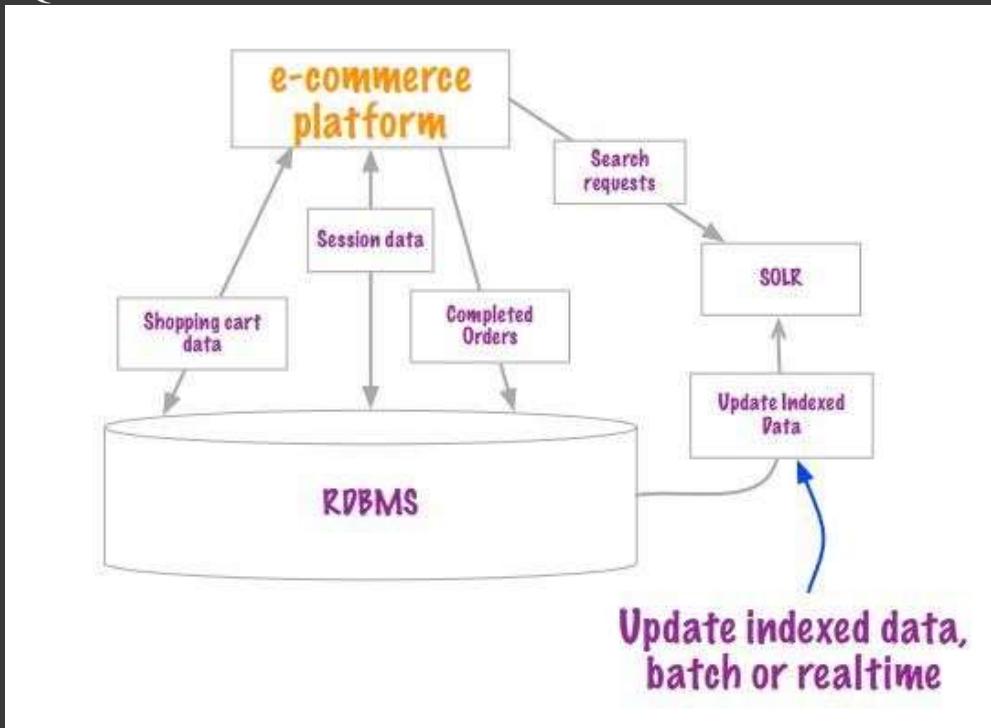
Mapping thuật ngữ trong Relational Database và NoSQL database

	Key-Value Relational (Riak)	Document (MongoDB)	Column-Family (Cassandra)	Graph (Neo4j)
instance	cluster	mongod	cluster	instance
table	bucket	collection	column-family	label
row	key-value	document	row	node
row-id	key	_id		
schema		database		schema

Bảng này chỉ có tính tương đối. Các node trong Graph DB không có key, mà các properties của nó sẽ đóng vai trò thay thế key.

Xu thế trong tương lai – Polyglot persistance

Như mình đã phân tích ở bài trước, NoSQL DB sẽ không thể nào thay thế được hoàn toàn RDBMS. Ngày trước, toàn bộ dữ liệu của một ứng dụng đều được lưu trữ dưới SQL Database.



Giờ đây, NoSQL sẽ dần chiếm vị trí của RDBMS. Một số chức năng của ứng dụng không quá coi trọng tính toàn vẹn của dữ liệu, nhưng lại cần tốc độ truy vấn nhanh, dễ truy vấn. Các chức năng này sẽ sử dụng NoSQL để truy vấn/lưu trữ dữ liệu.



Điều này tạo nên một xu hướng mới – áp dụng nhiều kiểu lưu trữ cho một ứng dụng, còn gọi là **polyglot persistence**. Các bạn đừng lo lắng kiến thức SQL của mình sẽ lỗi thời, vì **cái ngày NoSQL chiếm chỗ của RDBMS còn lâu lắm**. Tuy vậy, nếu các bạn muốn đi đầu về công nghệ, hoặc **tham gia các công ty start-up**, hãy tự trang bị cho mình kiến thức NoSQL nhé.

Bài viết có tham khảo từ sách [NoSQL Distilled](#) và bài viết của tác giả:<http://martinfowler.com/bliki/PolyglotPersistence.html>

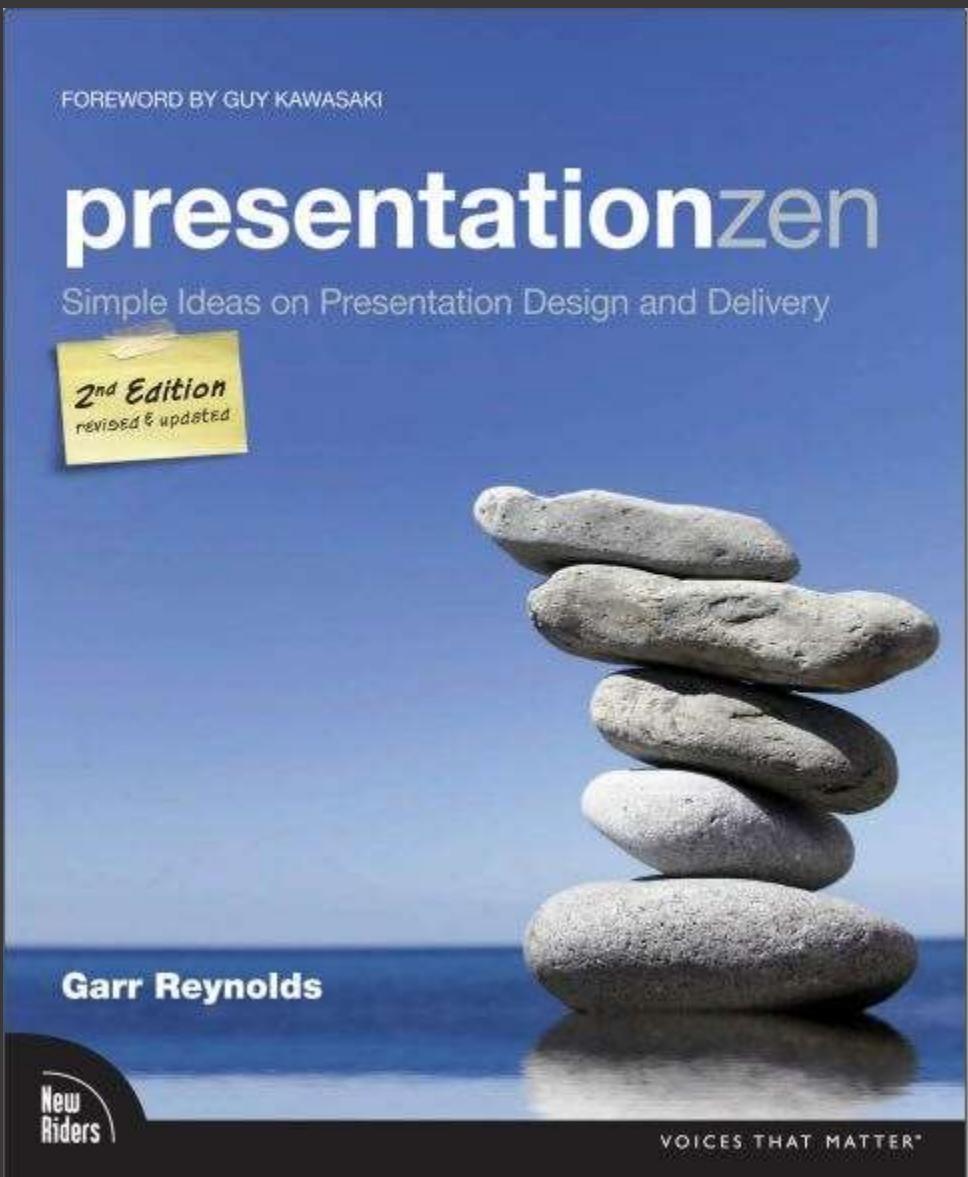
Review sách: Presentation Zen – 99% sinh viên/lập trình viên đã làm slide tệ hại như thế nào

Posted on 22/09/2015 by Pham Huy Hoàng

Thời sinh viên, thuyết trình luôn là nỗi ác mộng đối với mình và bạn bè: Phải chuẩn bị nội dung để nói, phân chia thời gian nói, và **làm slide**. Các bạn sinh viên nói chung (bao gồm cả mình) thường làm slide theo công thức đơn giản: Copy một đồng chữ từ đâu đó bỏ vào slide, thêm tiêu đề, thêm hình ảnh, căn chỉnh lại 1 tí, thế là đã có một slide hoàn chỉnh.

Đến thời đi làm, mình tưởng rằng đã thoát khỏi cái cảnh phải chuẩn bị slide thuyết trình. Thế nhưng, đến những lúc cần giới thiệu sản phẩm, công nghệ, hay khi tổ chức seminar, mình lại phải bật PowerPoint lên, làm lại công việc nhảm chán thời sinh viên. Hồi các anh senior, các bác Manager mới biết là muốn leo lên vị trí cao, muốn thăng tiến phải **giỏi thuyết trình, biết cách trình bày** thì mới được đồng nghiệp nể trọng, cấp trên chú ý.

Xem các hướng dẫn làm slide nhiều, mình cũng ngộ ra là: Slide tốt là slide ít chữ, nhiều hình, ngôn từ ngắn gọn. Đến một ngày, sau khi đọc xong 1 cuốn sách, mình ngộ ra được cái “đạo”, cái “tinh túy” của nghệ thuật làm slide, nghệ thuật thuyết trình, cuốn sách đó tên là ... **Presentation Zen**.



Giới thiệu nội dung

Tác giả là một người đam mê văn hóa Nhật. Trên một chuyến tàu điện, anh thấy một chàng trai đang hối hả sửa lại những slide chi chít chữ của mình. Nhìn xuống hộp bentou đang ăn dở, anh chợt nảy ra ý nghĩ “*Tại sao ta không tạo ra những slide đơn giản, thanh nhã, tinh tế như những hộp bentou*”. Đó là nguồn cảm hứng để anh áp dụng chất Thiền vào việc thuyết trình, thiết kế slide.

Theo thói quen thông thường, chúng ta hay nhét rất nhiều thông tin, số liệu vào slide. Hậu quả là slide dày đặc chi chít chữ, làm rối mắt khán giả. Sách giải thích rõ sai lầm cơ bản này:

- Slide là thứ khán giả nhìn và chú ý tới.

- Thông tin, số liệu sẽ được in ra dưới dạng handout và phát cho khán giả đọc, đừng nhét chúng vào slide.

Three Parts of a Presentation

I. Slides the audience will see
聞き手が見るスライド

Percentage of national land area covered by forests

Country	Percentage (%)
USA	30%
China	20%
Germany	15%
Sweden	40%
Japan	35%
Finland	70%

2. Notes only you will see
話す手が使うメモ

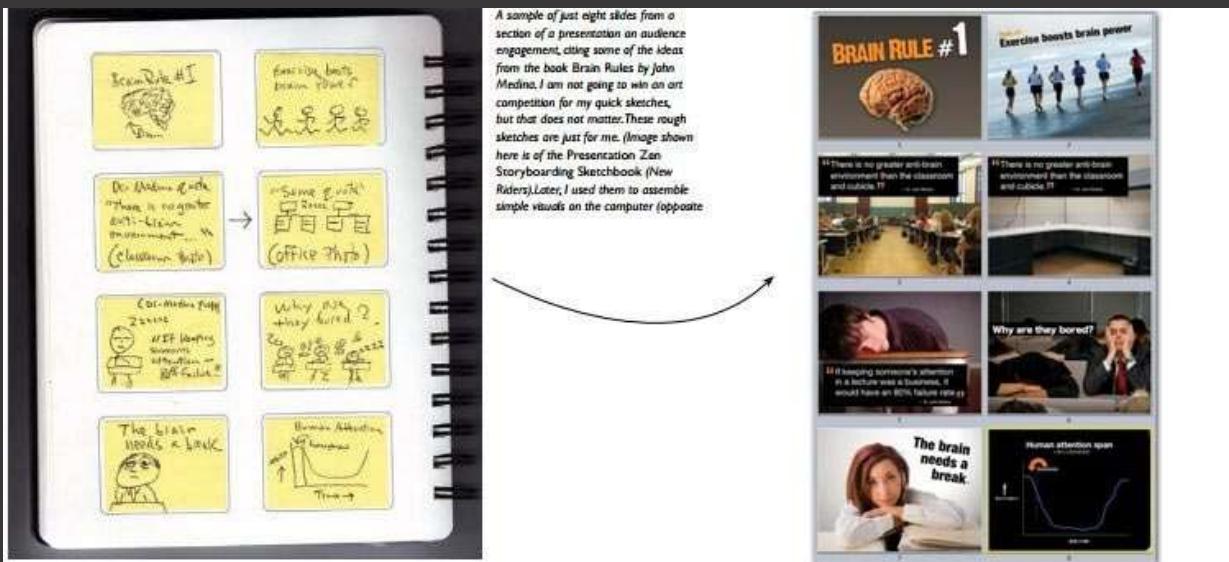
3. Handout to be taken away
講習のための配布資料

If you remember that there are three components to your presentation—the visuals, your notes, and the handout—then you will not feel the need to place so much information in your slides or other multimedia. Instead, you can place that information in your notes (for the purpose of rehearsing or as a backup) or in the handout. This point has been made by presentation experts such as Cliff Atkinson, yet most people still fill their slides with volumes of text and hard-to-see data and simply print out their slides instead of creating a separate leave-behind document.

(I have used the four slides on this page while making this point during my live talks on presentation design.)

Slides 聞き手が見るスライド
Notes 話す手が使うメモ
Handout 講習のための配布資料

Sách hướng dẫn cách chuẩn bị, sắp xếp nội dung slide, làm cho mọi thứ gắn kết với nhau. Mình áp dụng theo cách này và khá thành công: Vẽ phác nội dung của mỗi slide lên một tấm note nhỏ, dán lên sách hoặc bảng, mỗi slide là một note. Sau đó, ta có thể thay đổi, sửa chữa các note này để có bản thô, dựa vào đó để làm slide hoàn chỉnh.



Phần tiếp theo là phần hay nhất của cuốn sách. Tác giả giới thiệu những nguyên lý trong việc thiết kế slide và những sai lầm ta thường hay phạm phải. Mỗi nguyên lý đều đi kèm với những slide mẫu, giúp ta dễ dàng nhìn ra lỗi sai của mình. Mình đã phải thốt lên “ò, ra trước mình làm sai bét cả” mấy lần khi đọc tới phần này.

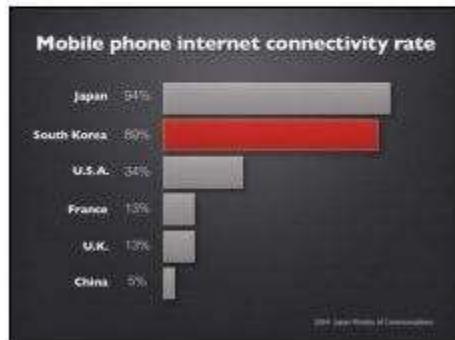
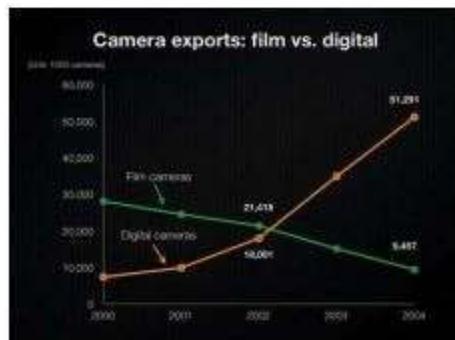
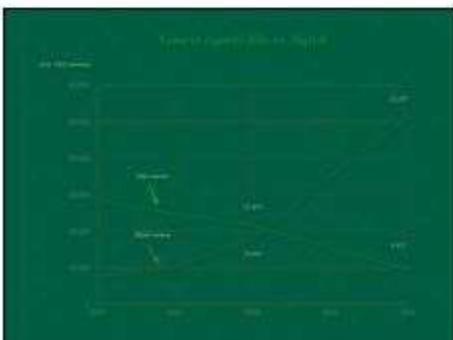


Lỗi cơ bản: Quá nhiều chi tiết thừa trong slide

WEAK CONTRAST



BETTER CONTRAST



Lỗi cơ bản: Slide thiếu tính tương phản, rối mắt

This title slide lacks a design priority. Due to the poor use of alignment and proximity, the slide seems to contain five different elements.

Principles of Presentation Design:

Tips on how to think like a designer

By Less Nessman

Director of the PRKW Institute



Principles of
Presentation Design

Tips on how to think like a designer

Less Nessman
Director PRKW Institute



This slide uses symmetrical balance and better proximity, with related items now clearly together. Greater contrast is achieved by adjusting the type size and color to give the design a clear priority.

Principles of Presentation Design

Tips on how to think like a designer

By Less Nessman

Director PRKW Institute



Principles of
Presentation Design

Tips on how to think like a designer

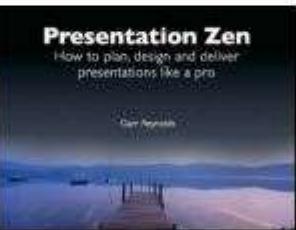
Less Nessman
Director PRKW Institute



Cách sử dụng font chữ, sắp xếp các đối tượng trong một slide



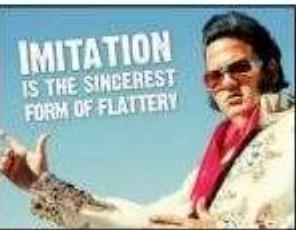
The slide on the left looks busier due to the abrupt contrast between the background color of the images. By aligning the text and photos and making the image backgrounds transparent (in this case, by simply changing the slide background to white) the slide is much cleaner and noise is reduced.



In the slide on the left, the background image has too much salience, making the title hard to see. Choosing a more appropriate background image that allows the text to remain clearly in the foreground and grouping the text lines makes for a stronger title slide.



By making the background of the fish photo seem transparent in the slide on the left, the image and text blend together harmoniously into a more unified visual. To make an image seem transparent, you can often simply change the slide background to white.



The slide on the left uses a busy template that makes the useful area of the slide about one-third smaller. The slide on the right covers the entire slide with the image. The text is clearly in the foreground while the image serves both as background and foreground, making the overall visual dynamic and unified with a cleaner, more dramatic look.

(Images on this page and opposite page from iStockphoto.com.)

Kết hợp hình và chữ: Hãy để hình ảnh đóng vai trò chủ đạo

This kind of slide is not unusual but suffers from typical problems such as a generic title, underlined text, and a small graphic that does not amplify the message of the presenter. Yellow text on this kind of blue background has been seen a million times before.



Here the same message as the slide on the left is made but the visual is big and bold and shows the trash problem in a way that illustrates and amplifies the presenter's story in a more visceral way. The type is gritty and the highlight color (green) subtly matches the tone of green of the plastic bottles.



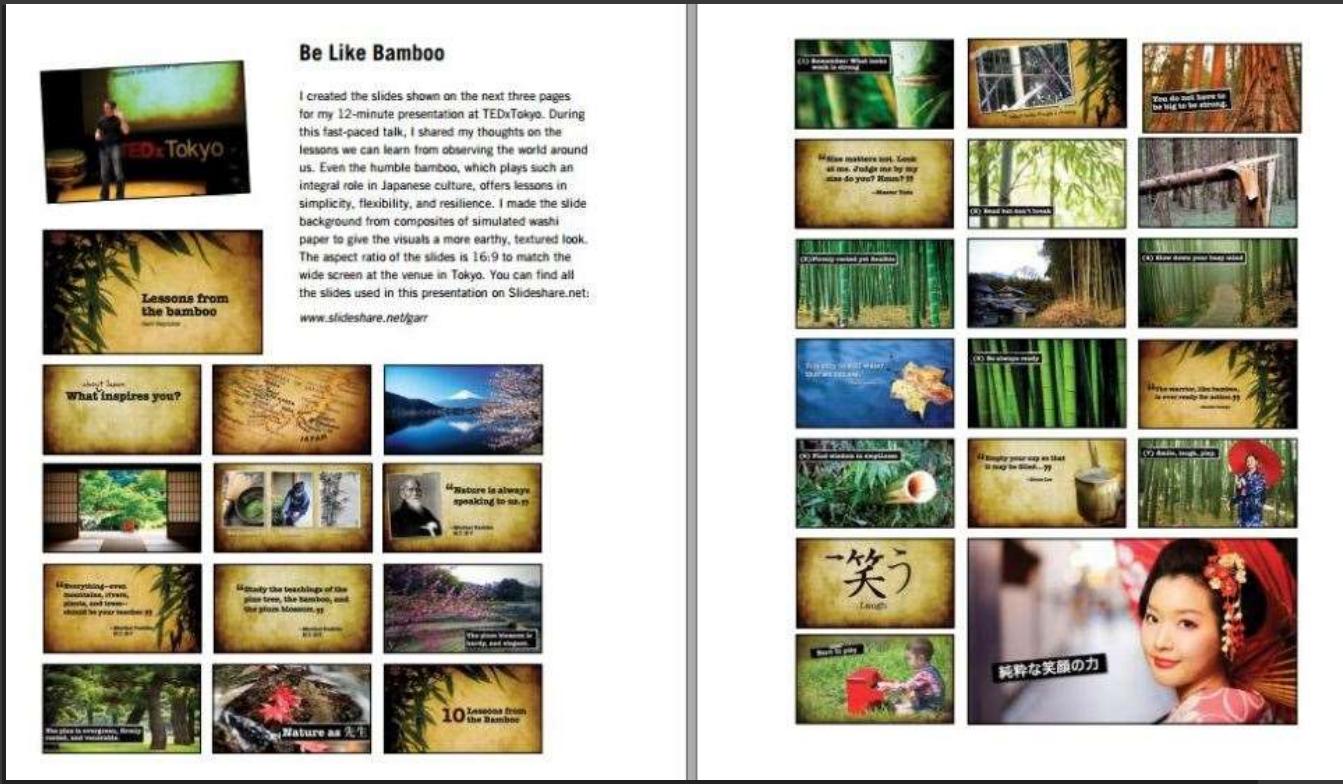
The quote and the photo are epic, yet this slide lacks impact or drama. The background looks very template-like and is too busy, making text hard to read. The type is a mess with the quote appearing as two bullet points. All elements are centered on the slide which results in trapped space. Even though there are few elements, the slide appears cluttered.



The type is clean and large. The photo is large and makes an impact, taking up the right third of the slide and bleeding off the edge. Distraacting background elements of the photo are removed. JFK's sight line is in the direction of the quote. Most viewers' eyes will go naturally to the face first and then smoothly to the type.

Biến slide “nửa mùa” thành slide “chuyên nghiệp” nhờ việc chọn hình ảnh, chọn font chữ

Tác giả đã rất râng rỗi khi tổng hợp khoảng vài chục slide mẫu vào sách để chúng ta học hỏi. Ta có thể cảm nhận chất Thiền, sự đơn giản, khúc chiết, gãy gọn của những slide này, từ đó rút ra kinh nghiệm co bản thân. Phần cuối cùng của sách hướng dẫn ta phong thái thuyết trình, cách kết nối với khán giả, làm buổi thuyết trình không buồn chán.



Một slide mẫu của tác giả

Nhận xét

Xuyên suốt bài viết, mình luôn dành những lời ngợi khen cho cuốn sách. Tác giả là một người giỏi thuyết trình, thích sử dụng hình ảnh. Cuốn sách của anh cũng tinh tế như một bài thuyết trình, với đầy đủ những hình ảnh ấn tượng, những ví dụ đơn giản, dễ hiểu dễ thấm.

Nhận xét cuối cùng: Đây là một cuốn sách **vô cùng đáng đọc**, đáng bỏ tiền ra mua. Những kiến thức trong sách sẽ có ích với hầu hết độc giả. Mong rằng các bạn cố gắng đọc và áp dụng, để những buổi thuyết trình trở nên hấp dẫn, lôi cuốn hơn, không còn là những buổi tra tấn người nói lẫn người nghe như hiện nay nữa.

Bonus 1: Một số sách hay cho các bạn lập trình

viên: <https://toidicodedao.wordpress.com/2015/04/21/lap-trinh-vien-trinh-cao-thi-nen-doc-sach-gi-phan-1/>

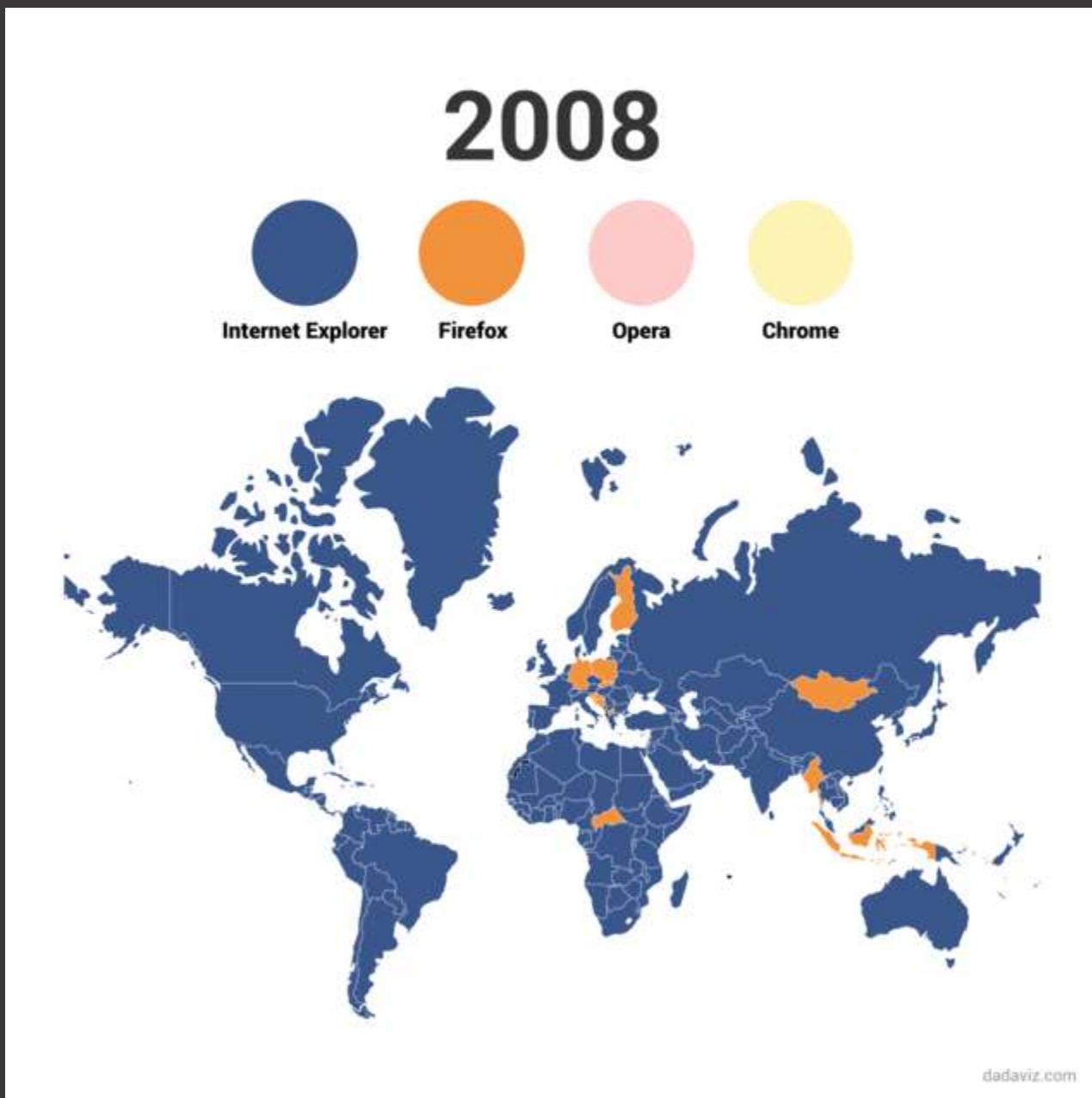
Bonus 2: Đây là một slide mình làm cho buổi thuyết trình về Dependency Injection và Inversion of Control trong công ty, có áp dụng một số nguyên lý trong sách. Mong các bạn góp ý và ủng hộ.

<http://www.slideshare.net/conanak99/ioc-and-mapper-in-c>

Tại sao giới lập trình thù ghét Internet Explorer?

Posted on 17/09/2015 by Phạm Huy Hoàng

Nhân dịp Microsoft ra mắt Window 10, khai tử IE và khai sinh trình duyệt Edge, mình viết bài này như một bài văn phúng điêu cho cuộc đời “lên voi xuống chó” của IE.



Xem biểu đồ, ta thấy thị phần của IE bị Firefox và Chrome cáp dàn. Có thật sự lý do chỉ là vì IE chậm, còn FF và Chrome nhanh và ổn định hơn? Thật ra, có một câu chuyện thú vị đằng sau sự căm thù của người dùng + developer dành cho IE đấy.

Ngày xưa ngày xưa, có 1 trình duyệt tuyệt dzời tên là IE...

IE bị mang tiếng là một trình duyệt chậm chạp, lạc hậu và nhiều lỗi. Thế nhưng không ai biết, nó từng có một quá khứ huy hoàng, mở đường cho bao nhiêu sáng tạo trên nền Web.

- **1996 – IE3:** IE3 là trình duyệt đầu tiên hỗ trợ CSS (Ghê chưa :o), ActiveX control, Java applet.
- **1997 – IE4:** Ra mắt một engine render siêu nhanh. Hỗ trợ HTML động, Javascript có thể xử lý, chỉnh sửa HTML. IE4 còn cross-platform, chạy được trên Mac OS, Solaris và Windows.
- **1999 – IE5:** Microsoft đã phát minh ra **Ajax** (Ngạc nhiên chưa :o). Phiên bản này đã giới thiệu XMLHttpRequest trong Javascript, công nghệ nền tảng cho Ajax ngày nay.



Giai đoạn 1996-1999, IE đã thổi một làn gió mới vào nền tảng Web. Để cạnh tranh với Netscape, các phiên bản IE ngày càng nhanh, nhiều chức năng và ổn định hơn, trở thành trình duyệt dẫn đầu thị trường. Thế rồi, đời ai học được chữ ngờ

Khi Microsoft ngủ quên trên chiến thắng

Một ngày đẹp trời nọ, Microsoft quyết định tích hợp IE làm trình duyệt mặc định cho Windows, nhằm hất cẳng Netscape khỏi cuộc chiến trình duyệt. Người dùng không

thể gỡ bỏ IE, cảm thấy Microsoft chơi xấu và dần dần ghét Microsoft. Tuy nhiên mọi chuyện chưa dừng lại ở đây.

- Ngủ quên trên chiến thắng: Năm 2001, với sự ra mắt của IE6, IE đã chiếm lĩnh 95% thị phần trình duyệt Web. Không còn đối thủ cạnh tranh, các lập trình viên IE chẳng còn gắng sức cải thiện và phát triển IE nữa (Thé mới buồn). IE không được nâng cấp gì trong vòng 5 năm, kể cả sau khi Firefox ra đời.
- Hậu quả của “sáng tạo”: Microsoft đã cố gắng bổ sung vô số chức năng và IE, cái giá phải trả là trình duyệt này không đáp ứng được nhiều chuẩn trong nền tảng Web. Khi IE còn độc bá thị trường, các chuẩn này không quan trọng lắm. Khi FF và Chrome nhảy vào cuộc chơi, điểm yếu của IE bắt đầu lộ rõ.
- Chậm chạp và cứng nhắc: IE6 bị bỏ bê với vô số lỗi cùng lỗ hổng bảo mật không được sửa. Mãi đến 6 năm sau, IE7 mới ra đời, thêm 1 số chức năng nhỏ, vẫn không đáp ứng chuẩn, và chỉ tạo thêm việc cho developer. Gần 3 năm sau, IE8 ra đời, có thể xem nó là một trình duyệt kha khá. Tuy nhiên, lúc này, đa số mọi người đã rục rịch chuyển qua FF hoặc Chrome hết cả rồi.



Lý do dân lập trình thù ghét IE

- Làm việc với IE là một cực hình: Hãy tưởng tượng, bạn phải code/design 1 trang web. Bạn bỏ 1 tiếng đồng hồ để test trên FF, Chrome, Safari, tất cả đều chạy ok. Bạn mở trang web đó trên IE6, IE7 và web “bể banh xác”. Bạn phải mất 1 tiếng để fix cho web chạy ok trên IE7, rồi mất thêm 1 tiếng nữa để fix trên IE6. Lý do đã nói phía trên, IE không tuân theo 1 số chuẩn của nền tảng web. Bạn phải tốn gấp 3 thời gian để hoàn thành công việc, vì 2 cái trình duyệt “khốn khiếp”. Là một lập trình viên, liệu bạn có “thích” IE nổi không?
- IE ở khắp mọi nơi: Cách đây mấy năm, hầu như các máy tính đều xài IE, một số trang web chỉ chạy được với IE. IE lại còn nằm chình ình trong Windows, không gỡ bỏ được. Việc bị ép buộc dùng IE mọi lúc mọi nơi làm cho IE “xưa đã bị ghét nay còn bị ghét hon”.



Tương lai nào cho IE

Mặc dù các bản IE10, IE11 chạy khá nhanh, tuân theo các chuẩn trên Web, IE vẫn bị người dùng thờ ơ lạnh nhạt. Trước tình cảnh này, Microsoft đã “thí quân giữ tướng”, khai tử IE và khai sinh trình duyệt Spartan (Sau này nó cũng chìm nghỉm luôn). Thị phần trình duyệt đã gần như bị Chrome và Firefox chiếm giữ.

Cũng phải khen lại Microsoft một tí. Với phiên bản Windows 10, Microsoft vừa giới thiệu trình duyệt Microsoft Edge mới, với lời quảng cáo “Nhanh hơn Chrome”. Mình có dùng thử thì thấy giao diện khá đẹp, **rất nhanh**. Hi vọng rằng Edge có thể lấy lại được hào quang đã mất của bậc đàn anh IE xưa kia.



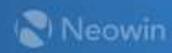
Internet Explorer



Project Spartan



Microsoft Edge



Bài viết có tham khảo từ nguồn: <http://www.howtogeek.com/howto/32372/htg-explains-why-do-so-many-geeks-hate-internet-explorer/>

Hướng dẫn tích hợp Travis-CI vào project trên github

Posted on 15/09/2015 by Phạm Huy Hoàng

Nhu mình giới thiệu ở bài trước, CI mang lại rất nhiều lợi ích cho một dự án phần mềm. Ở các công ty, sẽ có một server để cài đặt Team City, Hudson, Jenkin, TFS... phục vụ cho CI. Tuy nhiên, nếu các bạn làm bài tập nhóm, làm freelance hoặc project riêng, việc setup 1 server riêng cho CI khá là khó khăn.

May thay, với Travis-CI, mọi công việc cài đặt phức tạp đã được thực hiện sẵn cho bạn. Travis-CI có hỗ trợ tích hợp với github. Bạn chỉ cần config project github một chút là xong ngay. (Mình đã có 1 bài hướng dẫn tích hợp Visual Studio và github, nếu cần bạn có thể xem lại nhé).

The screenshot shows the Travis CI interface. On the left, there's a large banner with the text "Build Apps with Confidence" and "Focus on writing code. Let Travis CI take care of running your tests and deploying your apps." Below the banner is a cartoon character of a person wearing a hard hat and safety glasses, with the text "Travis CI". On the right, the main interface displays a list of repositories under "My Repositories". The first repository listed is "green-eggs/ham" with 3 builds, status passed, finished about 2 hours ago, and a commit message "use rvm install". Below this is "hop-on/pop" with 217 builds, status passed, finished about 3 hours ago. Further down are "one-fish/two-fish" with 542 builds, "fox/in/socks" with 2725 builds, and "yertle/turtle" with 282 builds. To the right of the repository list is a "Build Matrix" section showing four jobs: 3.1, 3.2, 3.3, and 3.4, each with their respective duration and finish time.

Cùng bắt đầu nào! Để test chức năng của CI, mình sẽ dùng 1 project có sử dụng Unit Test. Các bạn có thể sử dụng lại project "Viết Unit Test với NUnit" trong bài trước (Lưu ý là CI chạy trên nền Mono, chỉ chạy được NUnit, không chạy được MSTest đâu).

Nếu lười, các bạn có thể fork project của mình tại đây: <https://github.com/ToiDiCodeDaoSampleCode/travis-sample>. **Bắt buộc các bạn phải có tài khoản github.**

4 commits 1 branch 0 releases 1 contributor

branch: master + travis-sample / +

Create README.md

conanak99 authored a minute ago latest commit f0774dadd5

.nuget	Initial Project	12 minutes ago
TravisSample.Test	Initial Project	12 minutes ago
TravisSample	Initial Project	12 minutes ago
.gitattributes	Initial commit to add default .gitignore and .gitAttribute files.	14 minutes ago
.gitignore	Initial commit to add default .gitignore and .gitAttribute files.	14 minutes ago
.travis.yml	Add travis CI file	10 minutes ago
README.md	Create README.md	a minute ago
TravisSample.sln	Initial Project	12 minutes ago

README.md

travis-sample

This is a sample code to test the integration between Visual Studio and GitHub and Travis-CI.

Build status: build passing

Vào trang travis-ci.org, bấm “Sign in with Github” góc trên bên phải.

Travis CI Blog Status Help Sign in with GitHub

Test and Deploy with Confidence

Easily sync your GitHub projects with Travis CI and you'll be testing your code in minutes!

 Sign Up

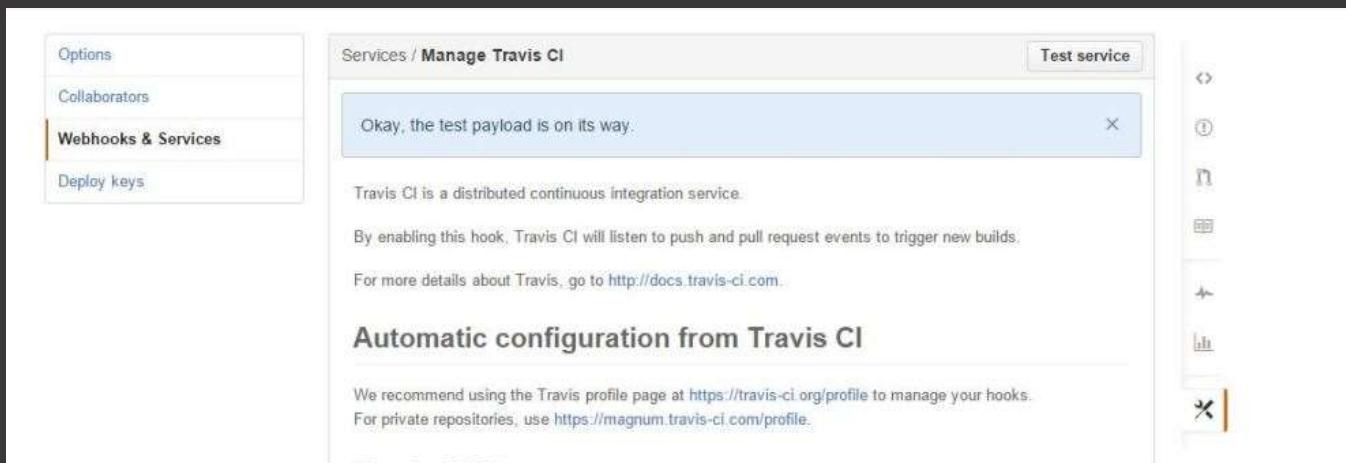
Sau khi đăng nhập, bấm vào dấu + bên trái, nhấp vào project muốn test để nó chuyển sang màu xanh là ok. Chờ một chút để Travis tích hợp vào project bên github.

The screenshot shows the GitHub profile page for user Phạm Huy Hoàng. At the top, it displays the user's name, profile picture, repository count (10), and a sync status indicating it was last synced about 7 hours ago. Below this, a message states: "We're only showing your public repositories. You can find your private projects on travis-ci.com." On the left, there's a section for "Organizations" with one listed: "ToiDiCodeDaoSampleCode" (Repositories 3). A note below asks if an organization is missing and provides a link to review and add authorized organizations. To the right, there are three numbered steps for setting up Travis CI: 1. "Flick the repository switch on" (with an icon of a switch being flicked), 2. "Add .travis.yml file to your repository" (with an icon of a file containing code), and 3. "Trigger your first build with a git push" (with an icon of a circular progress bar). Below these steps, there's a list of repositories with Travis CI status indicators: "ToiDiCodeDaoSampleCode/GitSample" (disabled), "ToiDiCodeDaoSampleCode/jasmine-sample" (enabled), and "ToiDiCodeDaoSampleCode/travis-sample" (enabled). A red arrow points to the "travis-sample" repository, which has a green checkmark indicating it is active.

Tiếp theo, bạn vào vào project trên github, kiểm tra xem nó đã nhận được Travis hay chưa?. Bấm Settings -> Webhooks & Services. Nếu không thấy TravisCI thì bấm Add Service, thêm TravisCI vào.

The screenshot shows the GitHub repository settings for 'travis-sample'. The top navigation bar includes 'Description', 'Website', 'Save or Cancel', and a 'Code' tab. Below this, repository statistics are displayed: 7 commits, 1 branch, 0 releases, and 1 contributor. The main content area shows a list of commits by 'conanak99' from July 9, with the latest commit being 'Initial Project' at 'master'. The sidebar on the left lists 'Options', 'Collaborators', 'Webhooks & Services' (which is highlighted with a red arrow), and 'Deploy keys'. The 'Settings' section includes fields for 'Repository name' (set to 'travis-sample'), 'Default branch' (set to 'master'), and a 'Rename' button. The 'Webhooks' section explains how external services can be notified via POST requests for events like pushes. The 'Services' section lists 'Travis CI' with a checked status, edit, and delete icons.

Bạn có thể bấm “Test service” để thử xem Github và Travis đã kết nối với nhau chưa.



Bước tiếp theo khá đơn giản, bạn chỉ cần tạo 1 file .travis.yml, commit lên thư mục trên github là xong. Bạn có thể tham khảo nội dung file .travis.yml trong folder của mình

language: csharp

solution: TravisSample.sln

install:

- nuget restore TravisSample.sln*
- nuget install NUnit.Runners -Version 2.6.4 -OutputDirectory testrunner*

script:

- xbuild /p:Configuration=Release TravisSample.sln*
- mono ./testrunner/NUnit.Runners.2.6.4/tools/nunit-console.exe ./TravisSample.Test/bin/Release/TravisSample.Test.dll*

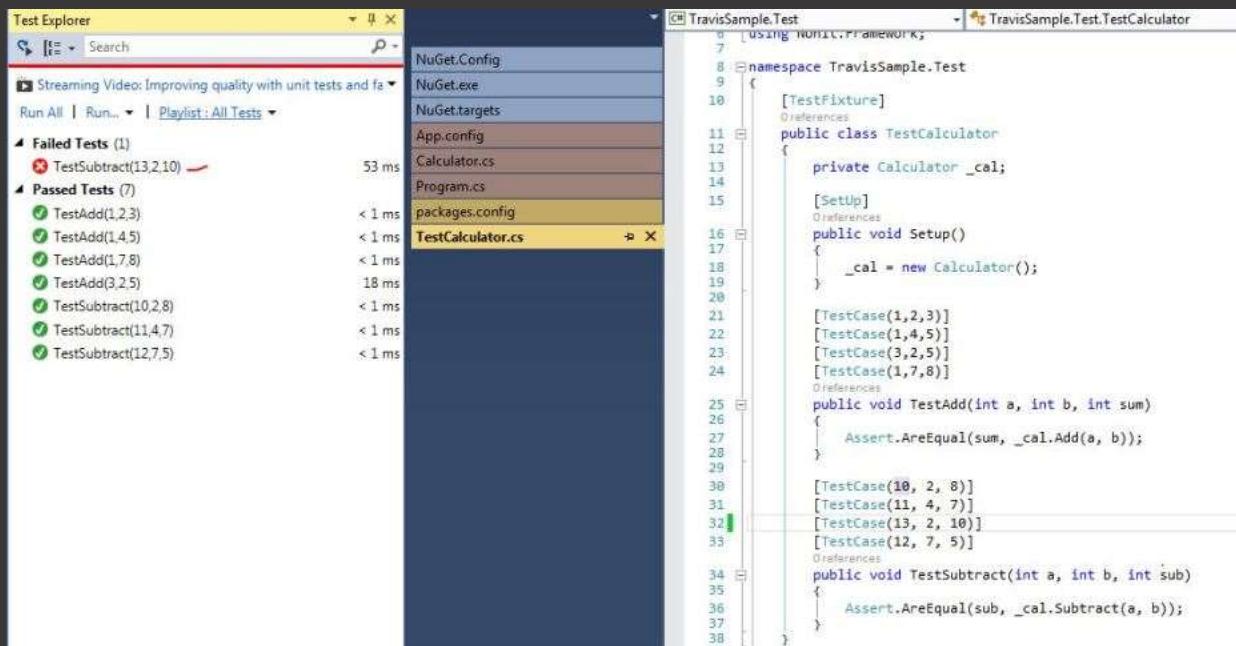
TravisCI sẽ thực hiện build dựa theo file .travis.yml này. Đầu tiên, nó sẽ restore các nuget package của solution. Sau đó build file `TravisSample.sln` và chạy các unit test. Ngay khi bạn vừa push lên git, bạn có thể vào web của Travis-CI để xem quá trình build bắt đầu chạy. Toàn bộ unit test cũng đã pass.

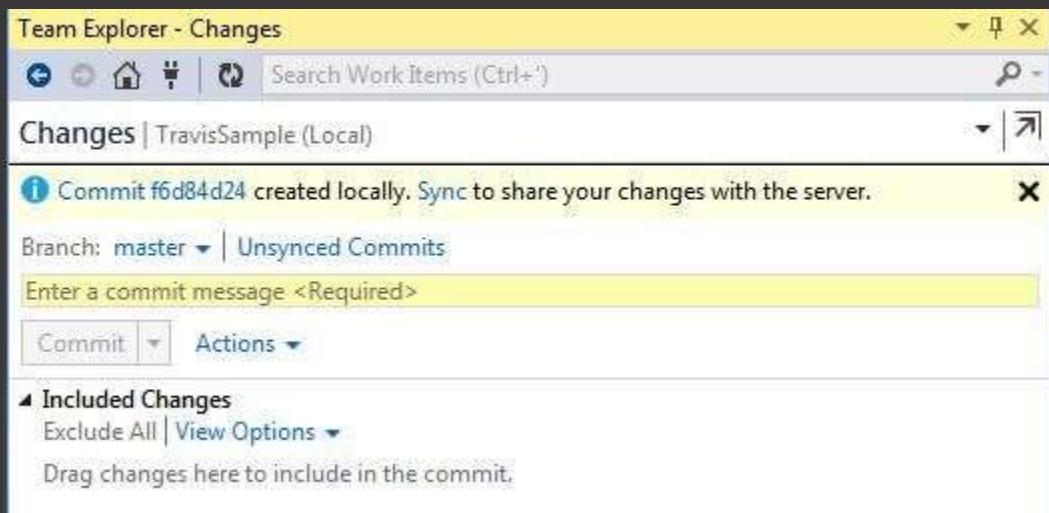
ToiDiCodeDaoSampleCode/travis-sample

build passing

```
1019 Build succeeded.  
1020     0 Warning(s)  
1021     0 Error(s)  
1022  
1023 Time Elapsed 00:00:01.2227280  
1024  
1025 The command "xbuild /p:Configuration=Release TravisSample.sln" exited with 0.  
1026 $ mono ./testrunner/NUnit.Runners.2.6.4/tools/nunit-console.exe ./TravisSample.Test/bin/Release/TravisSample.Test.dll  
1027 WARNING: The runtime version supported by this application is unavailable.  
1028 Using default runtime: v4.0.30319  
1029 NUnit-Console version 2.6.4.14350  
1030 Copyright (C) 2002-2012 Charlie Poole.  
1031 Copyright (C) 2002-2004 James W. Newkirk, Michael C. Two, Alexei A. Vorontsov.  
1032 Copyright (C) 2000-2002 Philip Craig.  
1033 All Rights Reserved.  
1034  
1035 Runtime Environment -  
1036   OS Version: Unix 3.13.0.40  
1037   CLR Version: 4.0.30319.17020 ( Mono 4.0 ( 4.0.2 (Stable 4.0.2.5/c99aa0c Wed Jun 24 10:04:37 UTC 2015) ) )  
1038  
1039 ProcessModel: Default DomainUsage: Single  
1040 Execution Runtime: mono-4.0  
1041 .....,  
1042 Tests run: 8, Errors: 0, Failures: 0, Inconclusive: 0, Time: 0.0414743 seconds  
1043 Not run: 0, Invalid: 0, Ignored: 0, Skipped: 0  
1044  
1045  
1046 The command "mono ./testrunner/NUnit.Runners.2.6.4/tools/nunit-console.exe ./TravisSample.Test/bin/Release/TravisSample.Test.dll" exited  
with 0.
```

Giả sử khi sửa code, bạn vô ý làm cho build bị lỗi, hoặc Unit Test chạy sai. Khi commit code lên, TravisCI sẽ tự động build, chạy unit test và báo lỗi.





Lỗi sẽ được báo ngay trên CI.

```
master Merge branch 'master' of https://github.com/ToiDiCodeDaoSampleCode/travis-5
conanak99 authored and committed

# 3 failed
Commit 851e848
Compare f0774da..851e848
ran for 1 min 14 sec
about a month ago

1842 Tests run: 8, Errors: 0, Failures: 1, Inconclusive: 0, Time: 0.0694089 seconds
1843 Not run: 0, Invalid: 0, Ignored: 0, Skipped: 0
1844
1845 Errors and Failures:
1846 1) Test Failure : TravisSample.Test.TestCalculator.TestSubtract(13,2,10)
1847     Expected: 10
1848     But was: 11
1849
1850 at TravisSample.Test.TestCalculator.TestSubtract (Int32 a, Int32 b, Int32 sub) [0x00000] in <filename unknown>:0
1851
1852
1853
1854 The command "mono ./testrunner/NUnit.Runners.2.6.4/tools/nunit-console.exe ./TravisSample.Test/bin/Release/TravisSample.Test.dll" exited
with 1.
```

Không chỉ thế, TravisCI còn gửi mail báo cáo về tình trạng bản build.

The screenshot shows an email from Travis CI regarding a GitHub repository named 'ToiDiCodeDaoSampleCode/travis-sample'. The email contains four log entries:

- Passed: ToiDiCodeDaoSampleCode/travis-sample#1 (master - cbcdf57)
- Errored: ToiDiCodeDaoSampleCode/travis-sample#2 (master - f0774da)
- Broken: ToiDiCodeDaoSampleCode/travis-sample#3 (master - 851e848)
- Fixed: ToiDiCodeDaoSampleCode/travis-sample#4 (master - c41fa7a)

The 'Broken' entry is highlighted. The email is from 'Travis CI <builds@travis-ci.org>' and is addressed to 'Me <huyhoang8a5@gmail.com>'. It was sent at 7/9/2015 1:46 PM. The subject is 'Broken: ToiDiCodeDaoSampleCode/travis-sample#3 (master - 851e848)'. The message body includes a link to the Travis CI build page for build #3, which shows a red 'Build #3 was broken.' status and a merge commit from 'conanak99'.

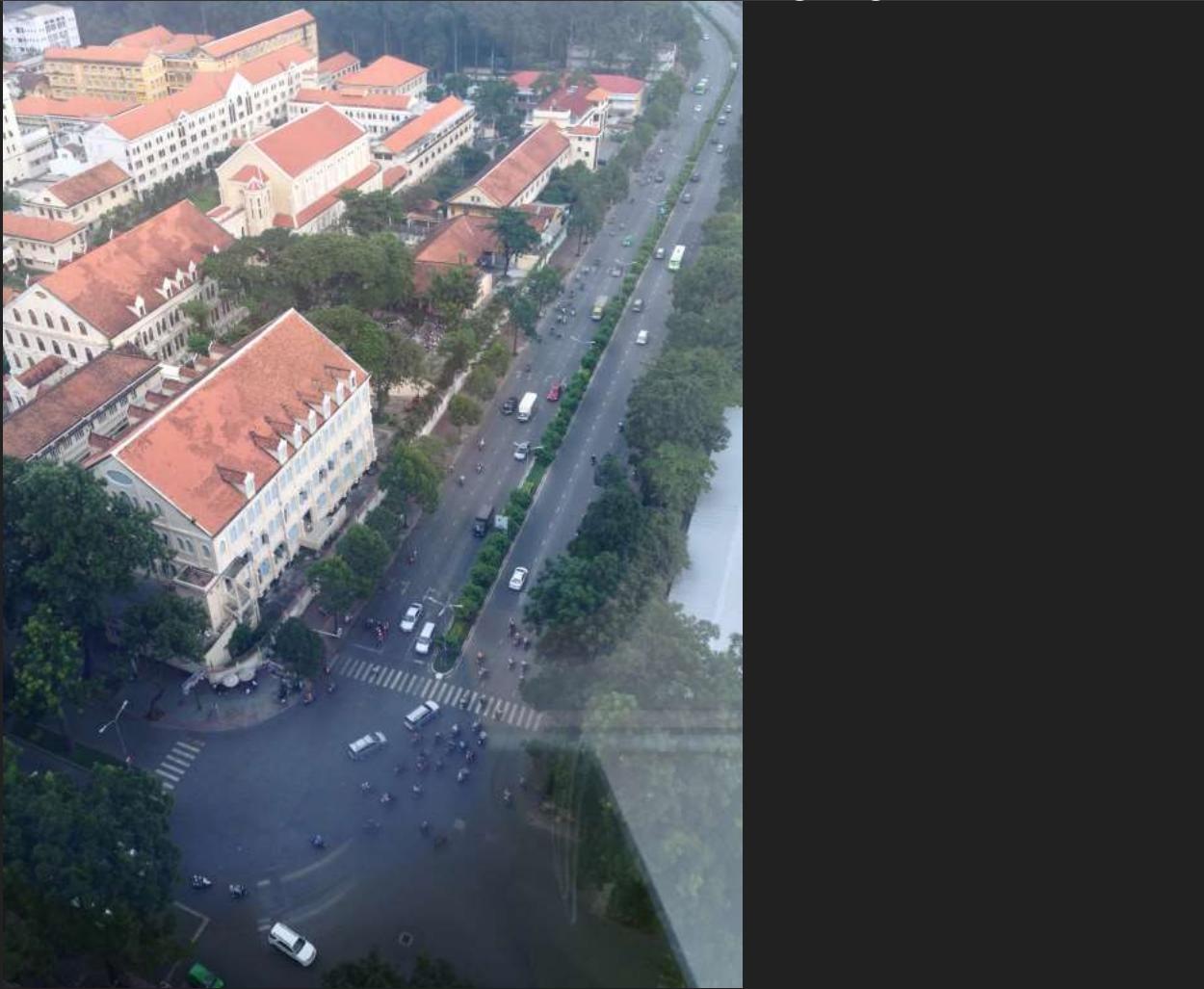
Không cần lấy code về, mình vẫn biết được code chạy được hay không, quá tiện lợi phải không nào? Với TravisCI, bạn có thể truy ra “thủ phạm” làm build bị lỗi, code chạy sai, không còn cảnh đùn đẩy đỡ thura qua lại khi code lỗi.

Qua bài viết, hi vọng các bạn hiểu thêm phần nào về công dụng cũng như cách áp dụng Continous Integration vào các dự án thực tế. Bạn nào có thắc mắc hay muốn tìm hiểu thêm về CI cứ để lại comment nhé.

Tạm biệt ASWIG – Đôi dòng tâm sự của chàng junior developer – Phần 1

Posted on 08/09/2015 by Phạm Huy Hoàng

Ngày 4 tháng 9 năm 2015, mình chính thức nghỉ việc tại Aswig Solutions để bước trên một con đường mới – *du học 2 năm ở UK tại Đại Học Lancaster*. Bài viết này là đôi dòng tâm sự kể về một năm làm việc của mình ở đây: Buồn ít, Vui nhiều, học được vô số điều bổ ích về technical và cách hành xử trong công việc.



Nghỉ việc rồi, không còn được vào phòng họp ngắm cảnh thế này nữa ~~

Cuối tháng 9/2014 – Bị phỏng vấn như super junior

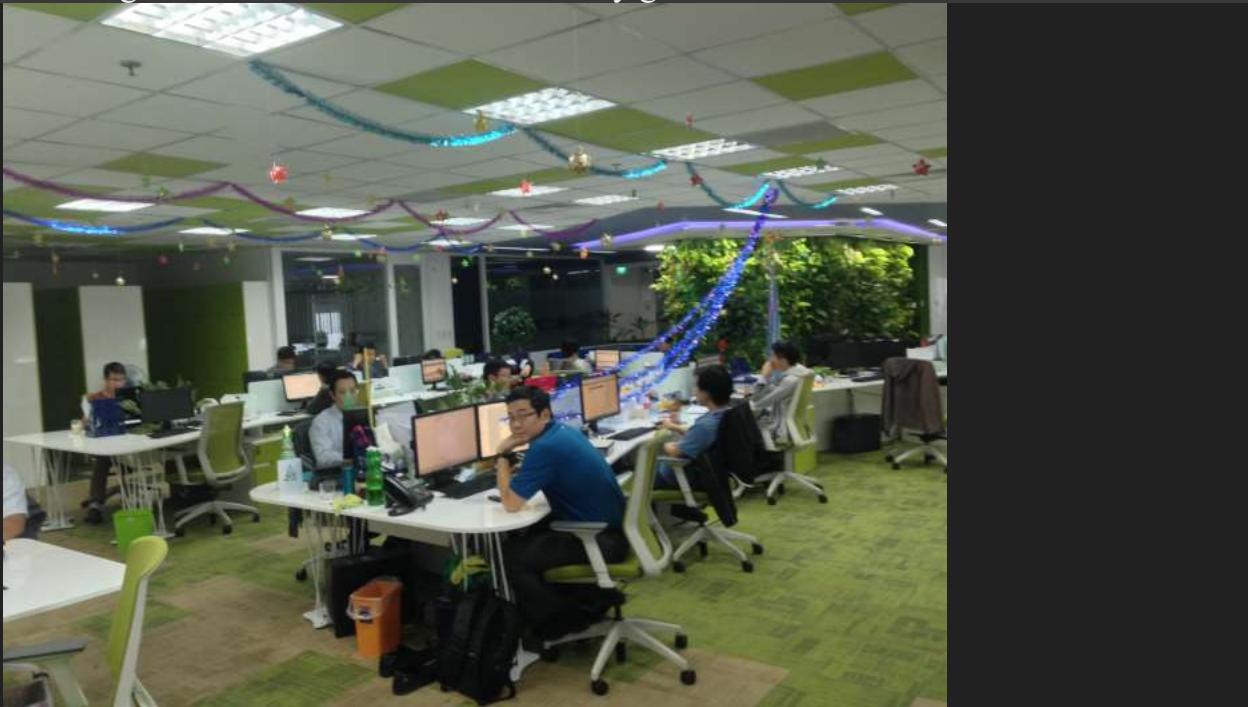
Nộp CV vào công ty với vị trí Junior Developer, mình “được” 2 ông anh team leader phỏng vấn tới tận hơn 1 tiếng rưỡi (100% tiếng Anh), junior mà hỏi đủ

thứ: C#, Linq, MVC, Entity Framework, Database, Front-end (HTML, CSS, Javascript), OOP, Design Pattern, ... riết rồi cứ tưởng tuyển siêu nhân chứ không phải junior gì hết.

Sau vòng 1 căng thẳng, mình được vào nói chuyện nhẹ nhàng với anh K. Manager (Trong lòng chắc mẩm là đã đậu). Đến bây giờ mình vẫn ấn tượng câu hỏi “**Em nghĩ 3 điều gì là quan trọng nhất đối với developer?**“. Sau này, mình mới biết vòng 2 là vòng đánh giá thái độ làm việc và tính cách, suýt nữa chém lung tung là rót rồi.

Sau khi deal lương, mình được hẹn phỏng vấn *thêm lần 3* với Technical Manager nước ngoài – *miếng ăn đến tận miệng còn chưa ăn được*, trời ạ =_= . Cũng may là mọi chuyện đều suôn sẻ, đầu tháng 10 mình bắt đầu đi làm, với mức lương khá cao so với mặt bằng junior nói chung (8 chữ số).

Công bằng mà nói, công ty khá là rộng rãi với nhân viên. Trong 2 tháng thử việc mình vẫn **được tính 100% lương**, còn được khám bệnh miễn phí. Cuối tháng 12 anh Manager vẫn xét thưởng tháng 13, được gần nửa tháng lương tiền thưởng :3 (Làm nguyên năm là được 2 tháng :o). Nghe đồn là anh Manager được thưởng nhiều quá, tiền đóng thuế đủ mua được cả 1 con xe tay ga...



Văn phòng xì tin và phong cách

Đầu tháng 10/2015 – Chập chững vào công ty, bị ma cũ “ăn hiếp”

Ngày đầu tiên đi làm, mình được anh Phóng – một anh cùng team hướng dẫn cài đặt máy, chỉ chỗ ăn này nọ (Công ty có hệ thống buddy, người mới gia nhập sẽ được “buddy” hướng dẫn và giới thiệu). Đến trưa, mình được anh H. Team Leader dắt đi ăn trưa (free). Tính mình dễ lầm, ai bao mình ăn là mình thương liền à.

Quy định của công ty là phải sử dụng tiếng Anh trong công việc. Mới đầu mình cũng hoang ngộp, vì thấy vài ông du học về hoặc phát âm như gió, lúc họp hành thì chẳng nghe được khách hàng nói gì. Sau 2 tuần mình mới tập thành thói quen và thích nghi được.

Mình lại ngồi đối diện anh Manager nên không dám ho he nhiều. Ngày xưa đi làm hôm nào mình cũng xem webtretho, vozforum, kenh14... Từ hồi qua đây, sáng nào mình cũng chỉ dám

vào simpleprogrammer.com, stackoverflow.com, codeproject.com, toidicodedao.com, ...

Mình được assign vào team Foundation, một team chuyên lo việc authentication và authorization cho các ứng dụng khác của công ty. Phase 1 của dự án không thành công lắm, và anh H. đã phải có một quyết định đau thương và táo bạo: **Đập nguyên phase 1, làm phase 2 lại từ đầu**. Team ban đầu có 3 người, về sau thêm được 1 người nữa là 4. Nhờ khả năng kỹ thuật xuất chúng của các thành viên, và khả năng chém gió “thần sầu” của anh team leader, **project đã thành công mĩ mãn**.

Kể chút về anh team leader của mình. Anh tên Hung, gốc Hà Nội. Nghe đồn gia đình anh cũng thuộc hàng “có số” ở đây. Do anh dụ dỗ một bé tiểu thư con giang hồ ngoài đây, sống thử 4 năm mà không chịu cưới. Bị trùm giang hồ bắt cưới nhưng anh không chịu, trốn Hà Nội để vào HCM lập nghiệp. Mỗi lần ngồi bên bàn nhậu, anh lại kể về chiến tích ngày xưa của mình. Với *tài năng xuất sắc và vẻ ngoài sáng chóe*, anh đã “bị” đưa lên báo để quảng bá cho công ty: <http://hrinsider.vietnamworks.com/nhat-ky-mr-amsterdam-hay-la-cau-chuyen-cua-hung-o-aswig-solutions/>.



Anh team leader đẹp trai, đã có vợ nhưng nghe đồn lấy vợ để che mắt thiên hạ

Giữa tháng 4/2015 – Company trip, Thái Lan vầy gọi

Mỗi năm công ty tổ chức company trip 1 lần, hầu như lần nào cũng đi nước ngoài. Kì này cả công ty được qua Thái Lan chơi, ngắm voi săn tiền chuyên giới luôn thê. Ở Bangkok 3 ngày nên mình cũng không đi chơi được gì nhiều. Các bố trong công ty rủ nhau đi “mát xa tới Z” bên Thái cho biết, nhưng nghe giá gần 1 củ ruồi nên ông nào cũng tặc lưỡi: *Thôi để tiền đó về Việt Nam rồi đi.* Đến hôm cuối cùng, mấy ông trong team rủ nhau đi mua quà cho vợ, ông nào ông nấy xách nguyên vali vài chục kg về. Ngẫm lại mới thấy có vợ cũng chẳng sung sướng gì.

Chuyện bên lề: Nhờ chuyến company trip này mà mình dụ dỗ được một bạn nữ khá xinh bên bộ phận Kế Toán của công ty. Chuyện riêng tư nên nói vậy cho các bạn thèm thôi :-“.



Dàn trai xinh gái đẹp của công ty, mà hầu như là “bóng đã có chậu” cả

Đầu tháng 5/2015 – Buồn thay chuyện kể ở người đi

Sau chuyến đi Thái Lan, nhờ khả năng ngoại giao xuất chúng, anh H. đã lấy một dự án khá béo bở về cho team, mang tên Claimbook. Anh Sơn – thành viên thứ 4 của team lại quyết định nghỉ việc, qua Singapore làm... rửa chén cho 1 phòng lab ở bên đó (Nghe đồn lương cao đến hơn 4.000 đô Sing).

Anh Sơn là **cựu sinh viên – trợ giảng trường Bách Khoa**. Khả năng technical của anh cũng rất khá, nhưng lúc nào anh cũng bảo: Cõi anh chỉ là cắc ké ở BK thôi. Mình nhủ thầm: *Cắc ké ở BK mà cõi này, dân BK chắc ghê gớm lắm.* Tới lúc thấy ông show *bằng Giỏi* mới biết mình bị lừa, hóa ra trên đài không tin ai cho được =_= Thiếu anh Sơn, đội ăn-nhậu-mát-xa mất đi một thành viên cộm cán, ai cũng buồn rười rượi.... [Còn tiếp phần 2](#).



Team Foundation ngày ấy (Anh đứng giữa nhảy vào chụp ké)

Tạm biệt ASWIG – Đôi dòng tâm sự của chàng junior developer – Phần 2

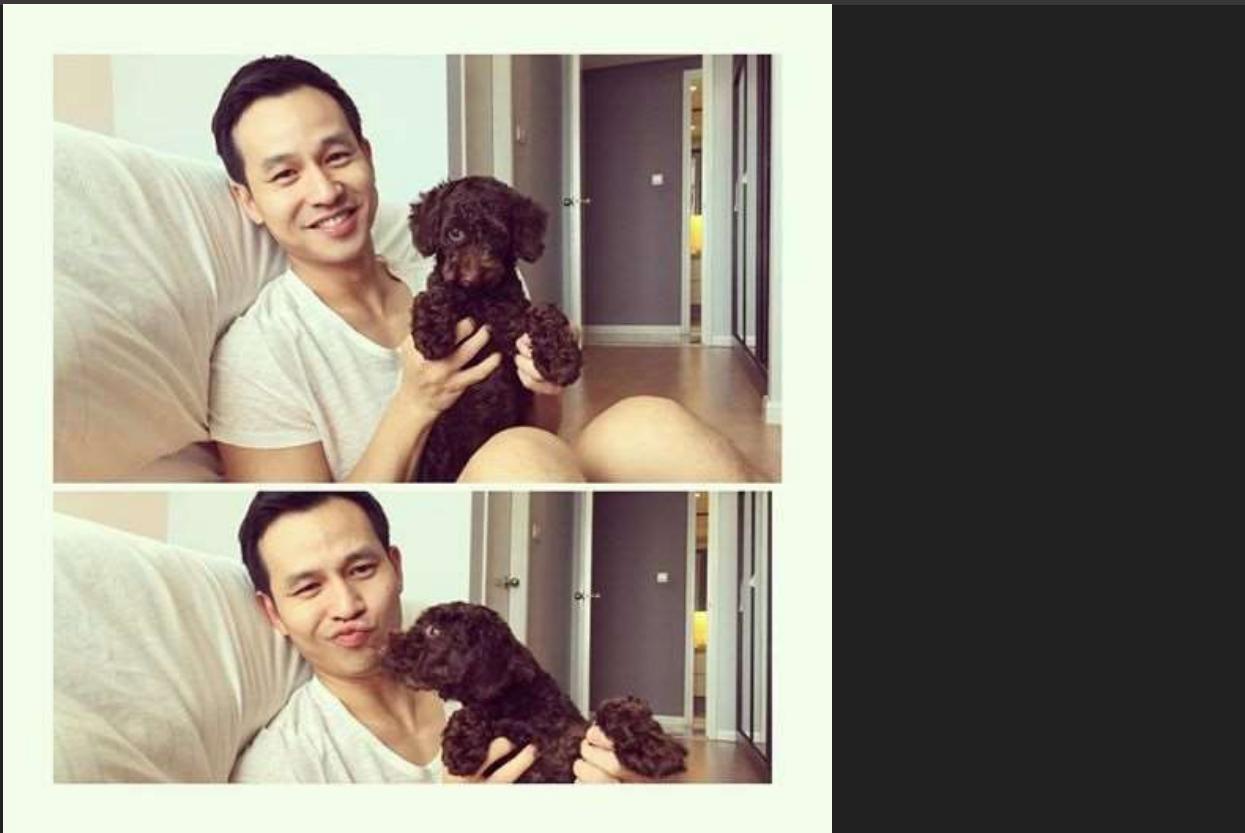
Posted on 10/09/2015 by Phạm Huy Hoàng

Đầu tháng 6/2015 – Một tháng đầy biến động

Tiếp theo [phần 1](#). Thiếu người, mình bị “ép” phải dựng toàn bộ wireframe, giao diện cho dự án Claimbook. Đúng là làm *super junior* nó khổ thế. Một lần nữa, anh H. lại tỏa sáng với khả năng ngoại giao của mình. Anh đã “lôi kéo” được một bác Technical Lead và 2 developer mới vào team. Anh còn dụ dỗ được team designer tham gia vào quá trình thiết kế cho dự án. Leader Team Designer là **anh Sơn Đoàn** – người yêu của Adrian Anh Tuấn. *Trông anh Sơn đẹp trai và manly hơn ông Team Leader của mình cả chục lần.*

Suốt 2 tháng trời, cả nhóm dồn tâm sức làm prototype cho dự án Claimbook. Team cũng tổn thất khá nhiều. Anh technical lead do thái độ làm việc không ok, không phù hợp với văn hóa công ty, vào team mình được 1 tháng đã phải chuyển qua team khác, rồi nghỉ việc ngay sau đó vài tuần. Design của dự án cũng bị thay đổi xoành xoạch theo yêu cầu của khách hàng, team mình cũng ngại không dám liên hệ lại với team designer vì sợ bị chửi do tự ý sửa lung tung design của họ.

Thay mặt anh H., xin gửi lời cảm ơn đến anh Sơn Đoàn đẹp trai, cùng toàn thể các bạn trong team designer đã góp phần vào thành công của dự án. Thấy team em được khen nhiều mà không nhắc đến các đóng góp của team anh nên cũng hơi cắn rứt.



Anh Sơn Đoàn – Giám đốc sáng tạo của ASWIG. Đẹp trai nam tính hơn ông team leader của mình nhiều T_T

Đầu tháng 8/2015 – Nghỉ việc, tăng lương và cách xử sự của người lãnh đạo
Vào khoảng tháng 5, mình cũng đã chia sẻ với anh Team Lead về ý định đi du học
vào cuối năm, để tiện cho việc sắp xếp nhân sự. Mặc dù khá tiếc nuối nhưng anh vẫn
ứng hộ lựa chọn của mình.

Cuối tháng 7, đầu tháng 8, công ty bắt đầu xét tăng lương. May mắn bạn mình ở
team khác được tăng cũng kha khá. Anh H. Team Lead khá khó xử khi phải giải thích
với mình rằng: Do mình sắp nghỉ, mình được thăng chức lên Software Engineer,
không được tăng lương, để phần đó lại cho các bạn. Cảm thấy vừa buồn vừa hụt hẫng,
mình viết một bức tâm thư gửi anh Manager và anh Team Leader (Thư dài nên mình
tóm tắt nhé).

*Chào hai anh, anh H. đã có chia sẻ với em về việc em được promote lên vị trí
Software Engineer và không tăng lương.*

Em rất hiểu và thông cảm lý do mà hai anh đưa ra quyết định này: Em cũng hiểu anh H. rất khó xử khi phải giải thích chuyện này cho em; anh K. chắc cũng phải đắn đo khi đưa ra quyết định này. Tuy nhiên, bản thân em cảm thấy rất buồn và thất vọng vì nhiều lý do sau: ...

Vì vậy, em mong hai anh có thể xem lại kết quả Performance Review của em, hoặc có cách gì компенсate cho em đỡ cảm thấy thiệt thòi. Cảm ơn hai anh vì đã đọc. Chúc hai anh có ngày cuối tuần vui vẻ.

Ngẫm lại, mình thấy mình kiềm chế cảm xúc khá tốt, không bù lu bù loa lên. Mình hiểu rằng ở vị trí Manager, có lúc phải đưa ra những quyết định thiệt thòi cho một cá nhân, để giữ lợi ích công ty và tập thể. Do đó mình cố gắng lịch sự khi viết thư, thử đặt mình vào vị trí của họ. Kinh nghiệm mình muốn chia sẻ là: *Quan hệ và thương hiệu bản thân là những thứ rất khó xây dựng, đừng vì cảm xúc nhất thời mà hành động nồng nỗi rồi làm mất cả hai.*



Thằng bạn nối khố (phải) – học chung lớp ở FPT, làm chung ASWIG, đãi mình đi ăn mừng tăng lương.

Ngay chiều hôm sau, mình được anh team lead và chị S. Manager rủ đi cafe nói chuyện, chia sẻ chân thành. Lần đầu mình được nhận lời xin lỗi chân thành của cấp

trên “vì đã quên consider suy nghĩ của em khi đưa ra quyết định“. Mình cũng được nghe chị S. Manager chia sẻ về việc các công ty VN đối xử không tốt với người sắp nghỉ. Bản thân công ty cũng đã có gắng thăng chức cho mình để khen ngợi những đóng góp của mình. Chỉ riêng việc **Team Leader và Manager chịu bỏ thời gian ra giải thích**, chia sẻ với junior như mình cũng làm mình cảm thấy được tôn trọng rất nhiều.

Tới hôm 1/9/2015, trong cuộc họp toàn công ty, mình được tuyên dương và tặng quà chia tay. Chưa bạn nào rời công ty lại có được đãi ngộ như vậy. Mình cũng hiểu là budget của công ty có hạn, anh H. Team Lead và anh K. Manager phải trích budget riêng để có quà cho mình, để mình cảm thấy được coi trọng, có thể nâng cao đầu tư hào khi rời khỏi công ty. Cảm ơn tình cảm và sự quan tâm mà 2 anh đã dành cho em. (Tai nghe AKG nghe sướng lắm các bạn ạ).



Nhin cái tai nghe chứ đừng nhìn mình, mình xấu trai lắm :D

Gần 1 năm đã qua, buồn vui cũng nhiều. Khả năng code của mình cũng tiến bộ đáng kể. Mình cũng quan sát và học hỏi được rất nhiều về cung cách lãnh đạo, quản lý nhờ quan sát anh H., anh K. và các anh team leader khác. Cảm ơn các anh đã giúp đỡ em trên quãng đường này. ASWIG, hẹn gặp lại.

P/S: Quảng cáo một chút: Công ty của mình đang tuyển nhiều vị trí như Dev/QA. Một năm được tận 20 ngày phép, 10 ngày nghỉ ốm, đi Company Trip nước ngoài. Thời gian làm việc flexible, ngày đủ 8 tiếng là được. Lương thường và các chính sách khác cũng rất khá. Không tin thì các bạn xem quảng cáo sẽ thấy: <http://hrinsider.vietnamworks.com/nhat-ky-mr-amsterdam-hay-la-cau-chuyen-cua-hung-o-aswig-solutions/>.

Bạn nào muốn gia nhập cứ gửi CV cho mình tại huyhoang8a5@gmail.com nhé, nếu được nhận mình sẽ chia hoa hồng cho.

Tổng quan về lập trình ứng dụng di động – Phần 1

Posted on 01/09/2015 by Phạm Huy Hoàng

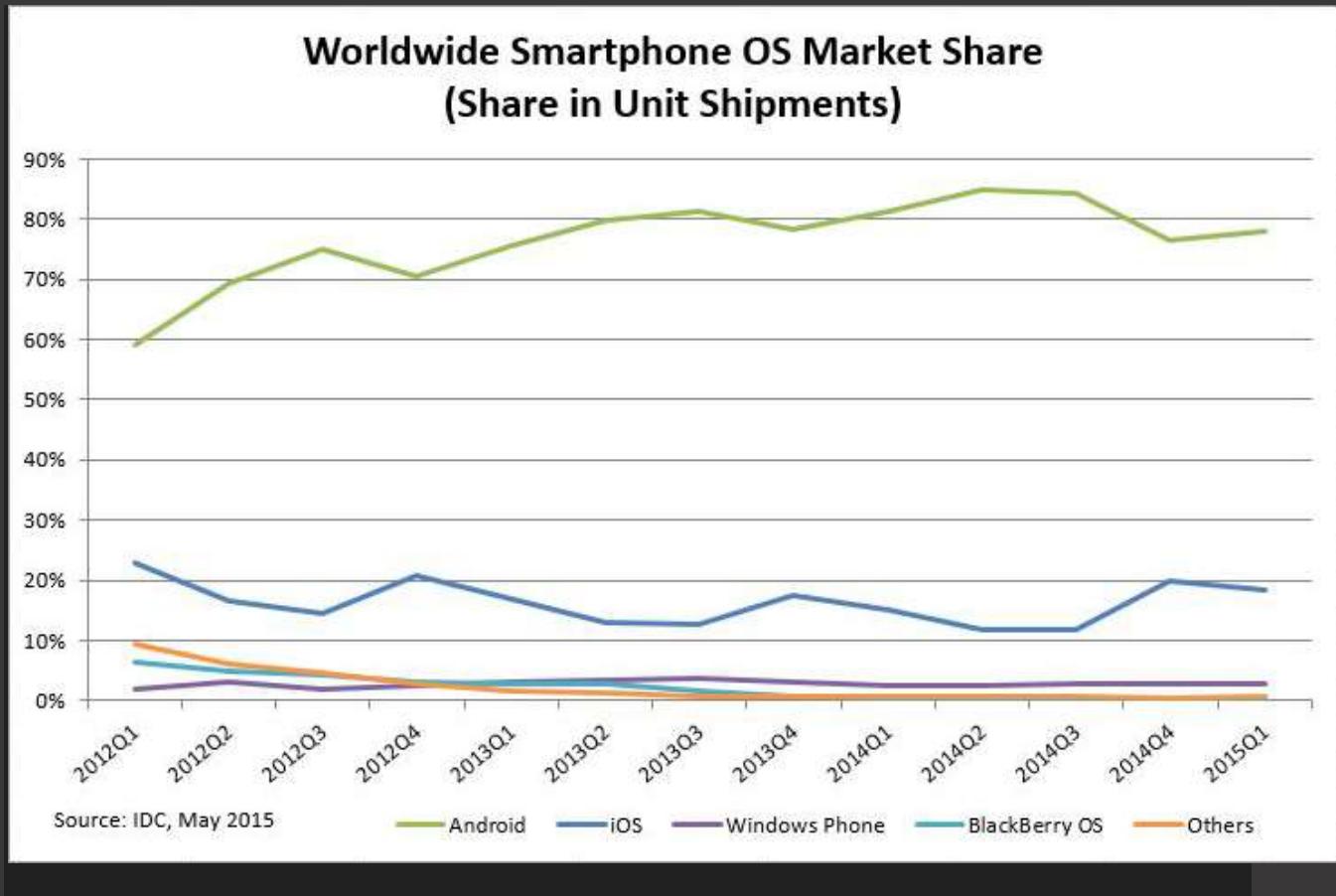
Trong khoảng thời gian gần đây, lập trình di động đang là một ngành hot. Các mẫu tin tuyển dụng gần nhất mình đọc thường tuyển Android developer, iOS developer, ... với mức lương khá cao, không thua kém gì lập trình web hay lập trình hệ thống nhúng. Ngoài ra, nếu biết cách lập trình ứng dụng, bạn cũng có thể làm freelance, hoặc tự phát triển ứng dụng và kiếm tiền thông qua ứng dụng của mình.

Bài viết này sẽ giúp các bạn có cái nhìn tổng quan về thị trường ứng dụng di động hiện nay, cũng như giới thiệu một số ngôn ngữ/công nghệ các bạn cần biết nếu muốn đi theo con đường này.



Thế chân vạc trong trận chiến Mobile

Trên thị trường ứng dụng di động hiện nay, 3 hệ điều hành chiếm thị phần cao nhất là : Android, iOS và Window phone, tiếp sau là 1 số hệ điều hành khác như BlackBerry... Trong phạm vi bài viết, mình chỉ phân tích về 3 OS đứng đầu là **Android, iOS và Windows Phone** nhé.



Thị phần trong thị trường di động của các hệ điều hành

Android – Kẻ chiếm miếng bánh lớn nhất

Theo biểu đồ, ta dễ nhận ra Android luôn **chiếm hơn 70% thị phần** của mảng di động. Ứng dụng Android được viết bằng **ngôn ngữ Java**, do đó các bạn lập trình viên Java có thể dễ dàng chuyển hướng qua mảng này.

Lập trình viên Android cũng đang là mục tiêu được các công ty săn đón. Các mẫu tin tuyển dụng Android developer **chiếm tỉ trọng lớn nhất** trong số các tin tuyển dụng của mảng mobile. Thuở còn làm đồ án tốt nghiệp, có 1 ông trong nhóm mình chưa biết gì về Android. Mình và ồng tự học và làm 2 tháng thực hiện đồ án, vừa xong đồ án thì ồng đi PV lập trình Android trong 1 công ty và được nhận luôn.

Android có quá nhiều device với đủ kích cỡ màn hình, cùng với vô số phiên bản (Từ 2.0 cho tới 4.4). Điều này gây khá nhiều khó khăn cho lập trình viên khi viết app : Cần phải test đủ thứ, đảm bảo ứng dụng tương thích với nhiều device, không bị lỗi giao diện, v...v.

Nếu bạn muốn đi theo con đường viết ứng dụng kiếm tiền, đưa ứng dụng lên Google Store, bạn sẽ phải mua 1 tài khoản Android Developer. Phí tài khoản này là **25\$/năm**.



iOS – Vị vua không ngai

Theo biểu đồ, iOS chỉ chiếm **20% thị phần, bằng 1/4 so với Android**. Tuy nhiên nghe đồn là doanh thu của Apple Store lại cao hơn Google Play Store. Nguyên nhân là do người dùng iOS chơi sang hơn, chịu khó bỏ tiền mua ứng dụng hơn so với người dùng Android.

Số lượng tuyển dụng iOS ít hơn Android, tuy nhiên lương cho lập trình viên iOS **lạinhỉnh hơn bên Android chút đỉnh**. Lý do không phải vì iOS tốt hơn Android, mà chỉ đơn thuần là qui luật cung cầu: Lập trình viên iOS hiếm hơn lập trình viên Android nên họ có giá cao hơn.

Để tiếp cận iOS, bạn cần máy ảo hoặc máy Mac để cài hệ điều hành MacOS. Ứng dụng iOS được viết bằng ngôn ngữ Objective-C (Giống C nhưng có thêm OOP) hoặc Swift. Việc code và debug trên iOS phức tạp hơn Android. Bạn phải cài đặt Xcode, **mua tài khoản Apple Developer** mới có thể test ứng dụng và đưa ứng dụng lên Apple Store. Bộ phận kiểm duyệt của Apple Store cũng khắt khe hơn Google Play Store, nhiều khi bạn phải chờ khá lâu để ứng dụng của mình được duyệt.

Nếu làm ở công ty, bạn sẽ được cấp tài khoản Apple Developer cũng như device để test. Nếu muốn tự viết, bạn sẽ phải tự trả **100\$/năm** cho tài khoản Apple Developer, và mất thêm 1 khoản kha khá để mua thiết bị (iPhone, iPad) về test.



Windows Phone – Kẻ sinh sau đẻ muộn

Windows Phone đã chậm chân khi gia nhập thị trường di động, nơi Android và iOS đã làm mưa làm gió khá lâu. Mặc dù Microsoft đã có một số chính sách hỗ trợ devloper, hệ thống ứng dụng trên Window App Store vẫn còn khá **nghèo nàn và nhảm chán**(Mình tìm app Google Map mà còn không có).

Thú thật, mình chả thấy công ty nào tuyển lập trình viên Windows Phone cả. Hầu như các công ty đều o bế cho ứng dụng trên Android, iOS trước rồi mới đến Windows Phone. Vì Windows Phone được viết bằng **ngôn ngữ C#** kết hợp với XAML, các lập trình viên C# có thể thử sức ở mảng này.

Cá nhân mình từng code cả Android lẫn Windows Phone thì thấy Windows Phone dễ code hơn, debug nhanh và tiện hơn. Với Android, nếu không có device, ta phải debug trên máy ảo, chạy rất chậm... máy ảo của Window Phone lại *rất mượt và nhanh*.

Nếu muốn viết app kiếm tiền, mình nghĩ các bạn nên thử Window Phone, vì những lý do sau (Nhớ cầu trời sau này Window App Store có nhiều người dùng hơn nhé, hiện tại ít người dùng nên chắc khả năng thu tiền lại cũng không cao đâu):

- Apple Store và Play Store đã có rất nhiều ứng dụng, tính cạnh tranh rất cao. Ngược lại, bạn ít khi gặp phải sự cạnh tranh trên Window Store.
- Microsoft đưa ra khá nhiều chính sách hỗ trợ Windows Phone, có thể trong tương lai sẽ thu hút nhiều người dùng hơn.
- Account Window Phone Developer có giá rất rẻ, chỉ có 19\$ và **dùng mãi mãi**.



Ở [phần 2](#), mình sẽ giới thiệu với các bạn những [ngôn ngữ/framework](#) dùng để viết 1 ứng dụng di động, cùng một số kĩ năng các bạn cần có nếu muốn theo con đường này, mong các bạn đón đọc.

Tổng quan về lập trình ứng dụng di động – Phần 2

Posted on [03/09/2015](#) by [Pham Huy Hoàng](#)

Ở [bài viết trước](#), mình đã giới thiệu tổng quát về [các hệ điều hành di động nổi tiếng](#) hiện nay. Trong bài viết này, mình sẽ đưa ra một số hướng phát triển ứng dụng, cùng với những ưu nhược điểm của nó.

Hiện nay, có 3 hướng chính để phát triển một ứng dụng di động, đó là: **Web App**, **Native App** và **Hybrid App**. Mỗi hướng sẽ cần những kĩ năng riêng, có những ưu nhược điểm riêng, sẽ được nói rõ hơn bên dưới.



Web App

Hướng Mobile Web thường được áp dụng khi các bạn đã có sẵn một website đang hoạt động. Ta sẽ **tạo thêm 1 trang web riêng cho mobile**, sử dụng HTML, CSS, một số framework hỗ trợ mobile và responsive (Bootstrap, jQuery Mobile, Materialize).

Người dùng sẽ trang web dành cho mobile để dùng ứng dụng.

Các xử lý khác liên quan đến backend như database sẽ được thực hiện phía trên server. Với một số công nghệ như AngularJS, một trang web có thể **giống y hệt một ứng dụng di động thật sự**. Mình có viết 1 game đơn giản trên nền web, bạn hãy thử vào bằng di động, nó cũng khá giống 1 ứng dụng thật sự: <http://hoangphpetprojects.github.io/LadyboyChallenge/>

Ưu điểm

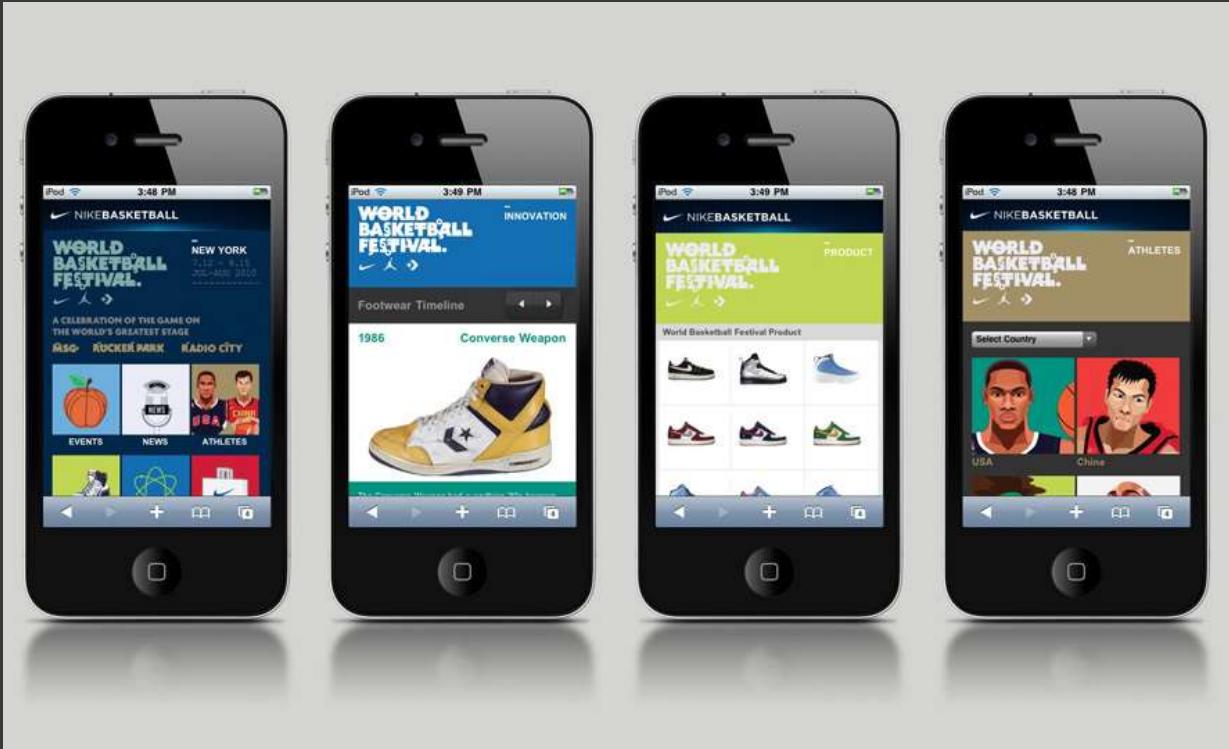
- Chỉ cần có kiến thức về web là viết được
- Viết một lần, chạy được trên mọi hệ điều hành
- Người dùng không cần phải cài app, có thể vào thẳng trang web
- Không cần phải thông qua App Store, **tiết kiệm tiền**
- Dễ nâng cấp (Chỉ việc nâng cấp web là xong)

Nhược điểm

- Với một số máy đời cũ, Web App sẽ bị bẻ giao diện, hiển thị sai, hoặc javascript không chạy.
- **Performance chậm**
- Không thể tận dụng được các tính năng của di động: Push notification, chụp hình, nghiêng máy, định vị GPS...

Kĩ năng cần có

- Kiến thức HTML, CSS, Javascript cơ bản.
- Kiến thức về một số framework responsive/mobile như: jQuery Mobile, Bootstrap, ...
- Một số framework javascript để viết Single Page Application: [AngularJS](#), [EmberJS](#), ...



Native App

Viết Native App nghĩa là lập trình viên **sẽ sử dụng IDE, SDK mà nhà sản xuất cung cấp** để lập trình ra một ứng dụng, build ứng dụng đó thành file cài và gửi lên App Store để kiểm duyệt. Người dùng sẽ phải tìm ứng dụng trên App Store, tải về máy và chạy.

Đây là hướng phát triển được áp dụng nhiều nhất, điển hình là game Flappy Bird của Nguyễn Hà Đông. Với những ứng dụng game, xử lý ảnh, cần tính toán nhiều, **Native App là lựa chọn duy nhất**.

Với những hệ thống lớn, cần đồng bộ, ta vẫn phải **viết phần back-end trên server**. Server sẽ đưa ra một số API. Native app lấy dữ liệu về máy, truyền dữ liệu lên server thông qua các API này.

Ưu điểm

- Tận dụng được **toàn bộ những tính năng của device**: Chụp ảnh, nghiêng máy, rung, GPS, notification.

- Có thể chạy được offline.
- Performance rất nhanh, vì code native sẽ được chạy trực tiếp.
- Là lựa chọn duy nhất cho các ứng dụng game, xử lý hình ảnh hay video ...

Khuyết điểm

- Cần cài đặt nặng nề (Eclipse, XCode, Android SDK, ...), khó tiếp cận.
- Với mỗi hệ điều hành, ta phải viết một ứng dụng riêng. **Khó đảm bảo sự đồng bộ giữa các ứng dụng** (1 button trên Android sẽ khác 1 button trên iOS, pop cũng khác).
- Cần phải **submit app lên App Store**, mỗi lần update phải thông báo người dùng.
- Code mệt và lâu hơn so với Mobile Web.

Kỹ năng cần có

- Ngôn ngữ lập trình: Java cho Android, Objective-C hoặc Swift cho iOS, C# cho Windows Phone.
- Kiến thức chuyên sâu về ứng dụng: View, Action, Adapter trong Android ...
- Cách xây dựng Web Service, Restful API, cách gọi API từ device, ...



Hybrid App

Hybrid App **kết hợp những ưu điểm** của Mobile Web và Native App. Ta xây dựng một ứng dụng bằng HTML, CSS, Javascript, chạy trên WebView của mobile. Tuy nhiên, Hybrid App vẫn có thể tận dụng những tính năng của device: chụp hình, GPS, rung,

Hybrid App sẽ được viết dựa trên một **cross-platform framework**: Cordova, Phonegap, Titanium, Ta sẽ gọi những chức năng của mobile thông qua API mà framework này cung cấp, dưới dạng Javascript. Bạn chỉ cần viết một lần, những framework này sẽ tự động dịch ứng dụng này ra các file cài đặt cho Android, iOS và Windows Phone.

Một số ứng dụng **không quá nặng về xử lý**, cần tận dụng chức năng của device sẽ chọn hướng phát triển này. Đây là một game dạng hybrid app mình viết bằng ionicPlatform, các bạn dùng Android có thể cài vào chơi thử nhé:<https://dl.dropboxusercontent.com/u/46157401/ladyboy-debug.apk>

Ưu điểm

- Chỉ cần biết HTML, CSS, JS (Thê nên mình mới khuyên các bạn nên học [Javascript](#)).
- Viết một lần, chạy được trên nhiều hệ điều hành
- Tận dụng được các chức năng của device.

Khuyết điểm

- Không ổn định, **khó debug**. Framework sẽ dịch code của bạn thành code native, việc sửa lỗi ứng dụng khá khó vì bạn không biết code sẽ được dịch ra như thế nào.
- Performance chậm.
- Cần cài đặt nhiều thứ (Titanium, Cordova đều bắt phải cài đặt SDK này nọ thì mới build ứng dụng được).

Kiến thức cần biết

- HTML, CSS, Javascript cơ bản.
- Cách dùng một số framework CSS, Javascript: jQuery Mobile, Ionic Framework, AngularJS, Bootstrap, ...
- Kiến thức về các **cross-platform framework**: Titanium, Cordova, Phonegap.
- Cách xây dựng Web Service, Restful API, cách gọi API từ device, ... (Hybrid app cũng sẽ kết nối với server thông qua API như Native App).

Hybrid Mobile App Development



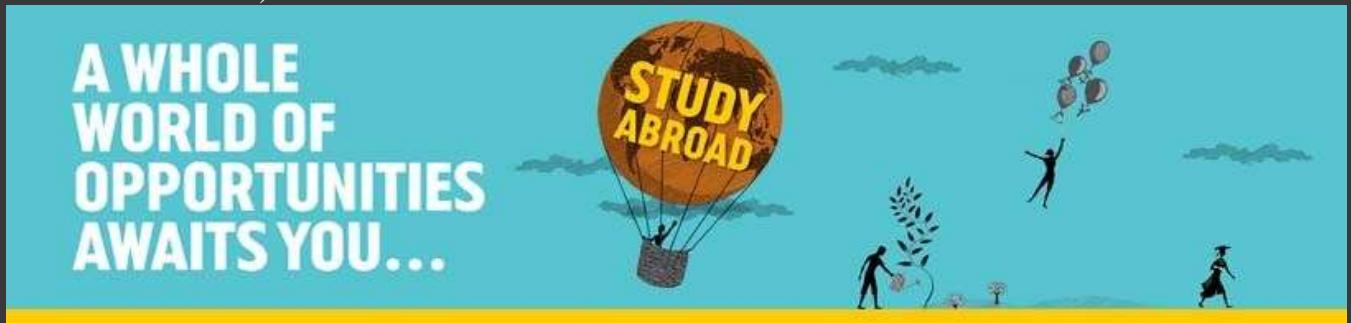
Sau 2 phần của bài viết, mình đã giúp các bạn đã có cái nhìn tổng quát về thị trường di động. Hi vọng các bạn có thể chọn được con đường mình muốn đi, cũng như biết được mình cần phải [học gì](#). Các bạn có thể đóng góp hay đưa ý kiến cho bài viết bằng cách comment nhé.

Tạm biệt Việt Nam – Chia sẻ kinh nghiệm nộp đơn du học – Phần 1

Posted on 27/10/2015 by Phạm Huy Hoàng

Bạn nào hay theo dõi blog của mình chắc cũng biết mình đã xách valy đi ngoài ra nước, nhầm, đi ra nước ngoài vào hôm 25/09. Sau gần hai năm đi làm, mình quyết định học lên Thạc Sĩ ở UK. Trong ngành IT, tấm bằng Thạc Sĩ không cải thiện được mức lương nhiều, nhưng mình thích đi để trải nghiệm. Bạn nào mong học Thạc Sĩ để lương cao hơn thì nên suy nghĩ lại nhe.

Sẵn có vài bạn hỏi nên mình viết bài để chia sẻ cách tìm trường và đi du học luôn. Quá trình tìm trường, nộp hồ sơ du học cũng khá nhiều khê và phức tạp. Nhờ sự giúp đỡ, tư vấn của trung tâm tư vấn nên mình được nhận vào Đại học Lancaster (Top 10 ở UK), với học bổng 1600 bảng (gần 550tr), ngành MSc Internation Innovation (Computer Science). Mình viết bài này chia sẻ kinh nghiệm chuẩn bị, nộp hồ sơ cho các bạn có dự định đi du học (Về kinh nghiệm học tập, ôn luyện tiếng Anh thì các bạn xem bài khác nhé).



Quy trình từ lúc nộp đơn tới nhập học như sau:

1. Chuẩn bị hồ sơ
2. Chọn trường, tìm ngành học.
3. Nộp hồ sơ.
4. Chờ offer letter, đóng tiền đặt cọc.
5. Lấy CAS, xin Visa.
6. Mua vé máy bay và lên đường.

Tổng thời gian sẽ mất khoảng 3-4 tháng. Vì các trường ở UK thường khai giảng vào tháng 9, các bạn nên bắt đầu lên kế hoạch vào tháng 4-5 là vừa. Phần một của bài viết sẽ hướng dẫn các bạn 3 bước đầu tiên trong quá trình làm thủ tục du học.

1. Chuẩn bị hồ sơ

Đây là những hồ sơ các bạn cần chuẩn bị để nộp:

- Bằng Đại học, bảng điểm. Nếu tiếng Việt thì phải nhờ dịch vụ dịch sang tiếng Anh.
- Passport.
- Bảng điểm IELTS. Yêu cầu tiếng Anh của ngành IT tựi mình không khó lắm, chỉ cần trên 6.5 overall, cả 4 phần nghe nói đọc viết không phần nào dưới 6.0.
- 2 thư giới thiệu của giáo viên và cấp trên (Nếu bạn chưa đi làm thì xin 2 thư giới thiệu của 2 giáo viên cũng được).
- Một thư giới thiệu bản thân, do bạn tự viết để giới thiệu mình, giải thích lý do bạn chọn trường này, ngành này, dự định trong tương lai. Thư dài khoảng 2000 chữ là ok.

Bạn nên **scan toàn bộ các giấy tờ trên** để tiện cho việc nộp online nhé. Thư giới thiệu thì bạn có thể gửi bản Word hoặc PDF, không cần in ra.

2. Chọn trường và tìm ngành học

Các bạn có thể vào địa chỉ: <http://www.thecompleteuniversityguide.co.uk/league-tables/rankings> để xem bảng xếp hạng các trường đại học ở UK (*Sao VN không có bảng xếp hạng nào như thế này nhỉ?*). Sau đó, *chọn ngành mình muốn học* bằng cách click vào link có tên ngành.

Chia sẻ thêm là mình thấy Lancaster rank 25 ngành Computer Sciences, mình định vào trường hạng thấp để học vừa sức, dễ làm trùm. Ai ngờ Overall Rank của nó cao quá xá, lỡ đâm lao đành phải theo lao vậy.

University League Table 2016

Subject	Year	Region	Group	Order by
Overall Score	2016	All	All	CUG Score
Subject	Overall Score	Accounting & Finance	Aeronautical & Manufacturing Engineering	
		Agriculture & Forestry	American Studies	Anatomy & Physiology
		Archaeology	Architecture	Art & Design
				Aural & Oral Sciences
				Biological Sciences
		Building	Business & Management Studies	Celtic Studies
				Chemical Engineering
		Chemistry	Civil Engineering	Classics & Ancient History
				Communication & Media Studies
				Complementary Medicine
				Computer Science
		Dentistry	Drama, Dance & Cinematics	East & South Asian Studies
				Economics
		Education	Electrical & Electronic Engineering	English
				Food Science
				French
		General Engineering	Geography & Environmental Science	Geology
				German
				History
		History of Art, Architecture & Design	Hospitality, Leisure, Recreation & Tourism	
		Iberian Languages	Italian	Land & Property Management
				Law
				Librarianship & Information Management
				Linguistics
				Marketing
				Materials Technology
		Mathematics	Mechanical Engineering	Medical Technology
				Medicine
		Middle Eastern & African Studies	Music	Nursing
				Ophthalmics
				Pharmacology & Pharmacy
				Philosophy
				Physics & Astronomy
				Physiotherapy
		Politics	Psychology	Russian & East European Languages
				Social Policy
		Social Work	Sociology	Sports Science
				Theology & Religious Studies
		Town & Country Planning and Landscape Design	Veterinary Medicine	

» If you cannot find your subject please check the [HESA subject listing](#).

Trên bảng xếp hạng, bạn có thể bấm vào link 1 trường để xem các thông tin liên quan tới trường đó (Điểm số, hình ảnh). Các bạn nên chọn trường dựa theo kết quả học tập ở Đại học. Nếu điểm tốt nghiệp và các môn cao, bạn có thể thử sức với các trường trong top 20. Với bảng điểm vừa phải, trung bình, việc vào các trường trong top 50 cũng không quá khó.

Bạn nên nhắm khoảng 5-10 trường mình thích, sau đó lên trang web của trường đó, tìm phần Postgraduate Study (Cao học). Như hình dưới là trang web của [University of Exeter](#).

The screenshot shows the homepage of the Exeter University Postgraduate Study website. The header features the university's logo and the text "POSTGRADUATE STUDY". Below the header is a navigation bar with links to Home, Taught subjects, Research subjects, Why study here, Apply, Money, International, Life, Careers, Visit, and Contact. A search bar is also present. The main content area includes a banner image of two students working on a laptop in a library, with the text "Applications for 2016 open Monday 5 October". To the right, there are three red callout boxes: "Find a degree" (with links to taught Masters by subject, research degrees by subject, and online application), "Find funding" (with links to PhD studentships, Masters scholarships, and the funding database), and "Find out more" (with links to a prospectus, about programmes, and open days). The left side of the page has a large graphic of a brick tower.

Bạn nên đọc kỹ *thông tin ngành học*, những gì sẽ được dạy, cấu trúc môn học thế nào. Nếu thấy hứng thú với những gì có trong chương trình thì hãy đăng ký học nhé. Hầu như các trường đều có mục **Apply Online**, cho phép bạn nộp hồ sơ nhập học online luôn. Nếu lười làm bạn có thể nhờ trung tâm làm hộ.

MSc IT Management for Business

Duration Full time 1 year
Discipline Computer Science
Location Exeter (Streatham)
Start date

Programme structure

Compulsory modules

Code	Module	Credits
BEMM114	Managing Operations	15
BEMM118	Strategic Innovation Management	15
BEMM119	Strategy	15
BEMM148	Marketing Strategy	15
ECMM417	Managing IT Projects	15

Mỗi trường có 1 số học bổng riêng, tùy theo chương trình hay ngành học. Các bạn có thể tìm trang web trường với các từ khóa funding, bursary, scholarship, ... Các học bổng này thường chỉ hỗ trợ 1 phần chi phí sinh hoạt, rất hiếm học bổng toàn phần nhé.

Lưu ý: Vào mùa tuyển sinh, các trường ở UK thường hay mở 1 số hội thảo du học ở VN. Nếu tham dự các hội thảo này, bạn có thể tìm hiểu về trường, trò chuyện với đại diện trường, cũng như nộp hồ sơ trực tiếp một cách nhanh chóng. Bạn hãy like page của một số trung tâm tư vấn du học trên facebook để có thêm thông tin nhé.

3. Nộp hồ sơ

Như đã nói, bạn có thể nộp hồ sơ online bằng cách vào trang web của trường, bấm vào chữ **Apply Online**. Một số thông tin bạn cần cung cấp khi nộp hồ sơ:

- Ngày tháng năm sinh, họ và tên.
- Passport (Tên, số passport, ngày cấp, nơi cấp) và bản scan.
- Chứng chỉ IELTS (Tên, mã số của chứng chỉ, ngày thi) và bản scan.
- Trường Đại học, ngành đã học, điểm tốt nghiệp.

- Kinh nghiệm làm việc nếu có (Mình khuyên các bạn nên đi làm rồi hẵng học, có gì về cũng đỡ lo thất nghiệp).
- CV (CV dạng học thuật, ghi những môn mình đã học), đừng nhầm với Resume xin việc nhé.
- Bản scan 2 thư giới thiệu (Có chữ ký). Nhớ phải có logo của trường, địa chỉ mail của cty hoặc trường nhé, không được dùng mail cá nhân. Một số trường còn gửi mail xác nhận cho người viết thư. Nếu bạn nhờ sếp hay giáo viên viết thư, nhớ dặn họ check email và xác nhận giùm.
- Personal Statement. Đây là bức thư bạn viết để giới thiệu bản thân: Những thứ bạn đã học, kinh nghiệm đi làm, lý do bạn chọn trường và ngành, dự định trong tương lai... Mình khuyên các bạn viết sẵn một thư làm mẫu, sau đó sửa lại một chút cho phù hợp với trường muốn nộp.



Những giấy tờ mình đã sử dụng khi nộp

Bạn nào cần mẫu thư giới thiệu, personal statement có thể tham khảo của mình: <https://www.dropbox.com/sh/cg2e8z8cu8m4wec/AABGqIzDKO9CasUWBXpCAusFa?dl=0>. Mình nộp 15 trường và được 13 trường nhận nên chắc cũng có chút giá trị tham khảo :D.

Các bạn đón xem [phần 2](#) để xem 3 bước còn lại nhé.

Tạm biệt Việt Nam – Chia sẻ kinh nghiệm nộp đơn du học – Phần 2

Posted on 29/10/2015 by Phạm Huy Hoàng

Nối tiếp [phần một](#), mình sẽ hướng dẫn các bạn 3 bước còn lại để hoàn tất thủ tục du học:

- Chờ offer letter, đóng tiền đặt cọc.
- Lấy CAS, xin Visa.
- Mua vé máy bay và lên đường.

4. Chờ offer letter, đóng tiền đặt cọc

Sau khi nộp đơn, bạn sẽ phải **chờ khoảng 2 tuần đến 1 tháng** để trường xét duyệt hồ sơ. Một vài trường sẽ tổ chức một buổi phỏng vấn ngắn (Qua skype hoặc điện thoại) để đánh giá bạn. Mình từng phỏng vấn với 1 ông bên Dundee hỏi về công việc, 1 ông bên York hỏi về kiến thức lập trình, 1 ông bên Lancaster phỏng vấn gần 1 tiếng để được nhận vào suất học bổng 16000 bảng. Đa phần các trường khác **chỉ xét hồ sơ**, nếu thấy ok sẽ gửi cho bạn offer letter mời nhập học.

Subject: MSc Unconditional Offer Letter attached -FPSE University of Southampton

From: Admissions at University of Southampton

- Congratulations on your offer!
- Offer of admission from the University of Aberdeen
- University of Dundee - Congratulations on your offer
- Newcastle University - Offer
- Your application to the University of Sheffield - Hoang Pham
- Your application to study at Newcastle University MSc Advanced Computer Science (full time)
- MSc (Taught): Advanced Computer Science (FT): Conditional Offer
- Offer of a place for MSc (Taught): Advanced Computer Science (FT) - University of Birmingham

From: Admissions at University of Southampton (MSc FPSE) <fpas-mscapply@soton.ac.uk>

Subject: MSc Unconditional Offer Letter attached -FPSE University of Southampton

To: Me <huyhoang8a5@gmail.com>

Please see the attached PDF File.

Dear Applicant,

Please find attached your official electronic unconditional offer letter. This letter can be used as evidence of your offer. It may also be viewed on your Web for Admissions account.

You should accept or decline this offer online via your Web for Admissions account.

This offer is valid for 30 days from the date of your offer, and it is essential that you reply within this period. If you have not received a response by this deadline, the offer will be withdrawn.

Consideration will be given to requests for extensions and reinstatement of applications after the deadline, subject to availability of places.

For information on University accommodation, please see: <http://www.soton.ac.uk/accommodation/>

INTERNATIONAL STUDENTS:

► 1 attachment: UOffer Letter-Southampton.pdf 172 KB

Sau khi nhận được offer letter, bạn có thể chờ 1-2 tháng để suy nghĩ có nên accept hay không. Nhiều trường chỉ cho bạn 1 tháng, nếu không accept offer sẽ tự hủy. Sau khi accept offer, bạn phải chuyển tiền đặt cọc (Khoảng 600-1000 bảng), số tiền này sẽ được trừ vào học phí.

5. Lấy CAS, xin Visa

Sau khi nhận tiền đặt cọc, trường sẽ gửi cho bạn 1 **thư xác nhận gọi là CAS** (Confirmation of Acceptance for Studies). Có thư này bạn mới có thể xin Visa được. Bạn cũng phải đóng tiền bảo hiểm NHS cho sinh viên (Khoảng 300 USD/năm) trước khi làm thủ tục xin Visa. Phí xin Visa hình như là khoảng 550 USD, ở HCM thì mắc hơn 50 USD, không hiểu vì sao.



Confirmation of Acceptance for Studies Details

Tier and Category

Tier and Category:

Tier 4 (General)

CAS Details

Sponsor Licence Number:	DVBF761XX
Sponsor Name:	Lancaster University
CAS Number:	E4G3XU2E20T0FX
Date Assigned:	08/06/2015
Sponsor Note:	

Student Details

Family Name:	Pham
Given Name(s):	Huy Hoang
Other Names:	
Date of Birth:	04/03/1992
Gender:	Male
Nationality:	Vietnam
Place of Birth:	
Country of Birth:	VIETNAM
Passport Number:	B2026811
Sponsor's Unique Student ID:	31703200

Course Details

Application Number:	31703200
Course Title:	MSc : International Innovation (Computer Science)
Course Level:	QCF/NQF7
Secondary Course Level:	
Course Start Date:	01/10/2015
Course End Date:	30/09/2017
Does the course require an Academic Technology Approval Scheme (ATAS) certificate?	No

Main Study Address in the United Kingdom

Address:	Lancaster University Bailrigg Lancaster Lancashire LA1 4YW
City or Town:	
County, Area, District or Province:	
Postcode:	

Evidence Provided

English Language Qualification

Page 1 of 2

Để xin Visa, bạn cần một số giấy tờ rắc rối khác nữa:

- Giấy khai sinh, sổ hộ khẩu.
- Giấy chứng minh số dư ngân hàng, giấy bảo lãnh nếu bố mẹ bạn đóng tiền.
- Bản gốc các giấy tờ nộp cho trường: Bằng ĐH, bằng IELTS, passport....

Sau khi làm hết thủ tục online, bạn cần mang toàn bộ giấy tờ đến trụ sở [VFS](#) để tiếp tục. Bạn cần thực hiện một cuộc phỏng vấn ngắn qua Lync lúc xin Visa, kết quả sẽ có sau 2 tuần. Đây là những câu hỏi thường được hỏi (kèm câu trả lời của mình):

1. Why did you choose UK to study?

Because UK universities rank among the best in the world and qualifications are internationally valued and recognized.

When I study in the UK I can meet people from different nationalities, sharing their backgrounds and discovering new perspectives.

2. Why did you choose this University?

Lancaster University is a highly-reputed school. It ranked No. 9 in the UK and No.1 in the North West by the Complete University Guide, and is in the top 1% of universities globally.

Moreover, I got a bursary of 16.000 pound from this school.

3. Why did you choose this major?

I studied Software Engineering at FPT University, so I want to learn more about Information Technologi at higher level.

The MSc International Innovation Computer Science pathway provides a unique blend of broad, contemporary academic modules and the professional training.

4. When is your course started?

On 1st October 2015

5. When you intend to travel to UK?

At the end of September, maybe on 21st or 23 September

6. Have you registered for HIS (immigration healthcare service)? If yes, when?

Not yet? (Cái này khi nào nộp vậy chỉ?)

5. What are the benefits from this major will bring to you or help you?

It can give me the practical knowledge and vocational skills I need as a software developer.

I also have the opportunity to practise the knowledge and skills I have learned through organised industrial placements and projects.

6. What is your plan in the future?

After finished the course, I will come back to Vietnam and work as a software developer, or open a start-up myself. It has been my dream since I graduated.

7. Where will be you stayed in the UK?

I will stay in Lancaster City. I plan to stay in the university's accomodation.

8. Do you have relatives in UK?

No

9. Do you understand all the questions I asked you?

Yes, I understand.

10. How long for your degree in UK this time, and which level?

Two years, level 7

11. How long did you get the offer after submitting the application?

More than one month, from the begining of April to late May

12. *What kind of documents you applied to the University (school) to get the offer?*

Degree, Academic transcript, IELTS, CV, Personal statement, 2 reference letters,

passport

13. How to accept the offer and pay the deposit?

I accept the offer by logging into the application website of the university, then click the accept offer.

After that, I got the payment information, and I went to Vietcombank to pay the deposit of 750 pounds

14. What is the most recent degree you have?

I got the bachelor degree of Software Engineering at FPT University in 2014

15. Did you look over Universities besides this one?

Yup, I got the unconditional offer from Southampton and Royal Holloway to.

16. What was the date you sent the application?

April 4th

Các giấy tờ của bạn sẽ bị VFS giữ khoảng 2 tuần. Sau đó các bạn sẽ nhận được 1 tin nhắn báo ra nhận lại Visa. Kết quả được niêm phong nên khi bạn mở phong bì ra, kiểm tra visa mới biết được. Visa ở Anh chỉ cho phép sinh viên đi làm thêm 20 tiếng/tuần nhé.



6. Mua vé máy bay và lên đường

Nếu muốn chắc chắn về chuyện chỗ ở, bạn có thể vào mục Accommodation của trường. Hầu như các trường đều có kí túc xá, giá hơi cao so với thuê ngoài nhưng đi lại thuận tiện hơn. Nếu có bạn bè hoặc người quen, bạn có thể qua sớm vài ngày, xin ở nhờ rồi từ từ tìm nhà, giá cả sẽ mềm hơn chút đỉnh.

Một trang web khá hay để săn vé máy bay giá rẻ là [skyscanner.net](#). Bạn chỉ cần điền một số thông tin cơ bản như: Ngày đi, điểm xuất phát (HCM). Ở Anh chỉ có 1 số sân bay quốc tế: London, Manchester, ... nếu thành phố bạn học không có sân bay quốc tế, hãy bay tới các sân bay này rồi đi tàu/xe về trường. Các trường đều có dịch vụ đưa đón tại sân bay, nhớ đăng ký trước nếu bạn sợ lạc đường.



Tới đây chắc các bạn cũng đã có đủ những thông tin cần thiết rồi. Bạn nào muốn hỏi thêm hoặc có kinh nghiệm hay muốn chia sẻ thì cứ comment vào bài viết nhé. Nếu muốn biết về cuộc sống học tập tại UK, bạn có thể xem [bài viết trước](#) của mình.

Ngồi xuống và viết blog đi nào

Posted on 20/10/2015 by Pham Huy Hoàng

Đây không phải là điều mình tự nói với mình đâu, mà là điều mình muốn nói với bạn đây. Mình nhắc lại một lần nữa nhé: Hãy ngồi xuống và viết blog đi nào.

Trước khi lắc đầu nguầy nguậy, đưa ra vô vàn lý do để chống chế: mình không có thời gian, biết gì mà viết, viết có được gì đâu ... Hãy chịu khó bỏ chút thời gian quý giá của bạn, đọc hết bài viết này rồi bắt tay vào viết nhé.



Viết blog có được lợi lộc gì không?

Dĩ nhiên là CÓ chứ. Viết blog là một chuyện mất nhiều thời gian và công sức, nếu như không được lợi lộc ai lại đi viết blog phải không nào. Một blog về IT sẽ mang cho bạn những lợi ích sau:

- Giúp bạn xây dựng và nâng cao thương hiệu bản thân: Mình nhận được sự tôn trọng từ mấy anh đồng nghiệp vì học thấy trên mạng và theo dõi blog của mình. Nếu bạn là senior, team leader, việc có 1 blog về IT sẽ làm đàn em nể nang, sếp coi trọng bạn hơn. Lúc xin việc, chỉ cần có CV trên linkedin, một account 2000 rep trên stackoverflow, 1 blog IT, bạn sẽ được đánh giá cao hơn hẳn các ứng viên khác nhé.

- Mở rộng mang lưới quan hệ: Nhờ việc viết blog mà mình làm quen được khá nhiều bạn, từ senior, junior cho tới các bạn sinh viên. Các mối quan hệ này khá hữu ích nếu bạn muốn chuyển việc, tham khảo mức lương.
- Nâng cao trình độ và khả năng viết lách: Muốn có một blog, dĩ nhiên là bạn phải viết. Bạn sẽ phải tự đốc thúc mình **tìm cái gì để viết, viết thế nào cho hay**. Trong quá trình đọc sách, tìm hiểu những thứ hay ho để viết, trình độ của bạn sẽ tăng lên đáng kể. Khi viết chúng ra, bạn sẽ nhớ rõ, nhớ lâu hơn, khả năng trình bày cũng càng ngày càng tiến bộ. Cái khả năng trình bày/chém gió/thuyết trình này quan trọng lắm đây nhé, nhiều khi nó quyết định vi trí – mức lương của bạn đấy.



Từ coder đến developer – Tôi đi code dạo

Lập trình viên giỏi không phải chỉ biết code

HOME

CHUYỆN CODING

CHUYỆN LINH TINH

CHUYỆN NGHỀ NGHIỆP

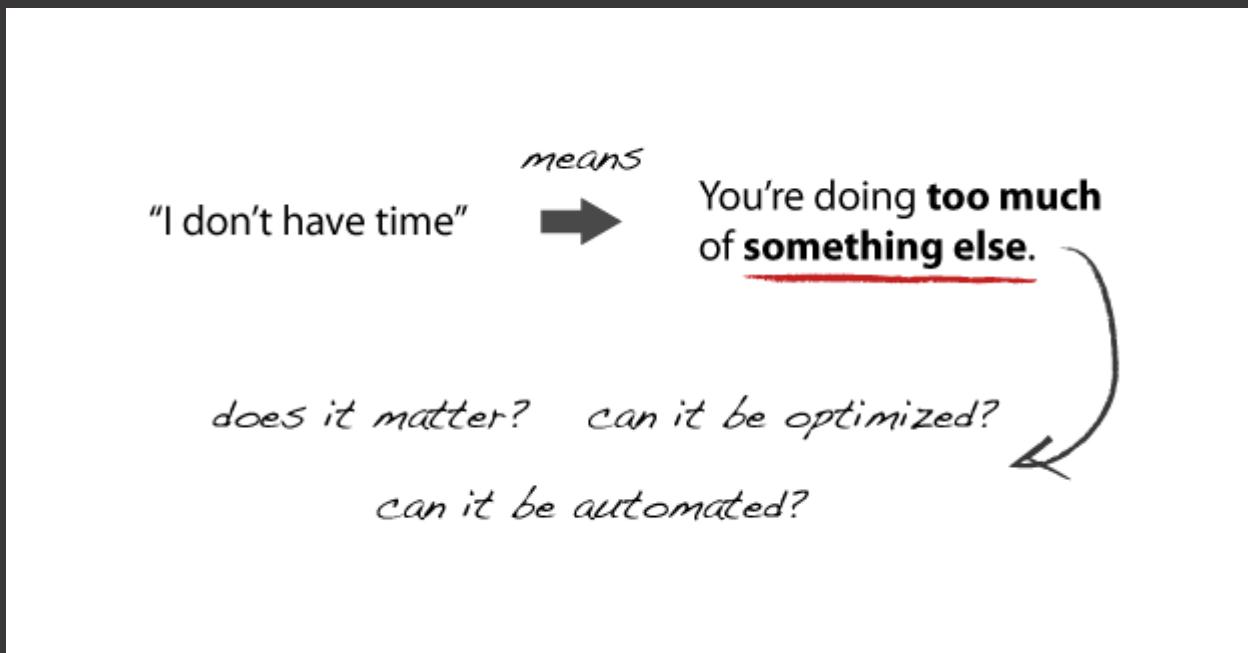
KIẾN THỨC CẨN THIẾT

ABOUT ME

Nghe có vẻ cũng hấp dẫn phải không. Chắc bạn cũng đang háo hức muốn bắt đầu, nhưng lại ngần ngại vì một số lý do dưới đây...

Thời gian rãnh rỗi đâu mà viết

Mình nhắc lại: **Viết blog là một chuyện khá mất thời gian**. Ngoài thời gian viết, bạn còn phải suy nghĩ đề tài, tìm hiểu về điều mình muốn viết, tìm hình minh họa. Thời gian cho một bài viết là khoảng 2-4 tiếng, chưa kể những bài có code demo phức tạp. Nếu chưa quen, bạn có thể viết 1 tuần/bài, siêng năng hơn thì 2 tuần/bài. Thử ngẫm lại xem, 4 tiếng/tuần không phải là 1 con số lớn: chỉ bằng 5 ván dota, 8 ván LoL, hoặc 1 buổi tối xem phim với gấu. Chỉ cần cắt tí thời gian Dota/LoL/Facebook/phim ảnh lại, bạn sẽ có 1 blog của riêng mình, được vô số những lợi ích mà mình đề cập phía trên.



Mình có biết gì đâu mà viết

Bạn nghĩ rằng mình chỉ là sinh viên năm 2, mới ra trường, biết gì đâu mà viết? Dễ thôi, hãy tìm hiểu những điểm hay của ngôn ngữ lập trình mình giỏi nhất, rồi chia sẻ nó với bạn bè.

Hoặc nếu bạn vừa học về OOP, SOLID, về SQL, hãy viết một post giải thích lại những điều mình học cho người khác, bạn sẽ nhớ và hiểu rõ hơn. Các cụ có câu “*Học thầy không tày học bạn*”, có thể các bạn SV khác sẽ thấy những điều bạn viết dễ hiểu, dễ thấm hơn so với bài giảng lê thê buồn ngủ của các thầy thì sao?

Bạn muốn viết một cái gì đó, nhưng người khác đã viết về nó rồi? Đừng lo, cứ viết, **bằng ngôn ngữ của chính bạn**. Có thể bạn sẽ tìm ra cách giải thích hay hơn, dễ hiểu hơn.

Bạn sợ mình chưa nắm rõ vấn đề, viết lung tung sẽ bị ném đá. Đừng lo, cộng đồng mạng rất là rãnh rỗi và sẵn lòng hỗ trợ cho bạn. Nếu bài viết của bạn có đôi chỗ sai sót, họ sẵn lòng bỏ thời gian giảng giải (ném đá) cho bạn. Hãy chia sẻ kiến thức (có thể sai) của mình để nhận lại kiến thức đúng đắn hơn nhé.

5. Kết bài của bạn lại đánh nhau đổ máu với mở bài. Mấy chục dòng trước đó bạn nhận định MEAN lần lượt và thay thế cho LAMP. Ôn trời là bạn đã nhận định được NoSQL ko thể thay thế cho SQL nhưng mình xin khẳng định thêm là 2 thứ giải quyết 2 vấn đề khác nhau. Như Tàu thủy ko thể nào mang đi thay thế cho Xe máy vậy.

Phần 2 mình mong có chút chuyên sâu hơn thì chỉ là giới thiệu cơ bản. Các cách mô tả của bạn cho thấy là bạn chưa đi sâu lắm, về bản chất ko có hàng hay cột nào cả, chỉ là một đám bytes được lưu trên disk và chúng được tổ chức khác nhau thế nào mới cho thấy được sự khác nhau giữa các loại DBMS. Còn nói về tên gọi cột hàng, bảng thì cassandra thay vì gọi là ColumnFamily mình gọi là table cũng được vậy 😊

Nói chung bạn nên hiểu một cách thấu đáo sâu sắc thì mới nên viết bài phân tích kéo lại gây ngộ nhận cho newbie. Còn nếu chỉ ở mức đã sử dụng qua thì hơi vội.

Một thanh niên đã đọc từng dòng code của Cassandra, 1 năm dùng MongoDB, 1 năm dùng Kyoto Cabinet, 1 năm dùng Redis, 3 năm dùng Memcached, 6 năm dùng MySQL chia sẻ, 2 tháng dùng ObjectDB 😂

Bỏ thích - Trả lời · 2 · 29 Tháng 9 lúc 18:14

 Huy Hoàng Phạm Cảm ơn góp ý của bạn 😊
Thích - Trả lời · 29 Tháng 9 lúc 18:21

Nghe hay đây, chỉ mình cách bắt đầu đi nào

Lập một blog vô cùng đơn giản. Bạn chỉ cần vào <https://wordpress.com/>, đăng kí một tài khoản, chọn một cái tên miền cho blog. Tiếp theo, bạn chỉ cần chọn themes, đặt một cái title thật kêu, đăng một cái banner thật ngầu và oách, thế là bạn đã có một [blog](#).

Tìm nội dung để viết cũng dễ thôi. Nếu bạn giỏi C, C++, hãy viết về nó, tài liệu tiếng Việt của mấy thằng này khá ít. Đọc một cuốn sách hay, hãy giới thiệu nó cho mọi người. Nghĩ ra cách học mới, hãy đăng lên để người khác vào chia sẻ. Điều quan trọng là phải giữ nhịp độ viết. Cuộc sống là một cuộc thi marathon, không phải thi chạy nước rút. Dừng viết nhiều khi sung, chán thì bỏ xó, hãy giữ nhịp độ 1,2 bài/tuần. Dần dần blog của bạn sẽ lớn dần, có nhiều nội dung hơn.

Giai đoạn đầu, blog của bạn sẽ vắng như chùa bà đanh, một ngày chỉ khoảng 5-10 lượt xem. Đừng lo, đây là giai đoạn để bạn tạo content cho blog, hãy cứ giữ [thói quen viết bài đều đặn](#), chịu khó quảng bá, dần dần mọi người sẽ biết tới blog của bạn thôi.



Những tháng đầu, cả tháng mình chỉ được khoảng 1-200 views, trong khi hiện tại một ngày mình được 500-800 view lận. Hãy tự đặt mục tiêu, tự thưởng cho mình khi blog của bạn đạt mốc [1000 view](#), [50 bài viết](#), ... bạn sẽ thấy hào hứng hơn nhiều, có khi còn vui hơn viết code ấy.

Vậy thì bạn còn chờ gì nữa, **ngồi xuống và bắt đầu viết blog thôi nào**.

Chuyện về các cây đa cây đề trong làng Software Engineering

Posted on 15/10/2015 by Phạm Huy Hoàng

Để thành một lập trình viên giỏi, có rất nhiều bạn phải học và phải biết: Cách viết code sach và refactor code, design thế nào để code SOLID, Inversion of Control và Dependency Injection, Agile methodology, ...

Tuy nhiên, đã bao giờ bạn tự hỏi: “Ai là người đã nghĩ ra những thứ đấy” chưa?. Bài viết này sẽ kể bạn nghe về những người đó. Đây là những cái tên có nhiều đóng góp to lớn cho ngành phần mềm. Họ nổi tiếng không chỉ nhờ khả năng code, mà còn nhờ **khả năng viết và diễn đạt, truyền cảm hứng**.



Ko chỉ là một developer, họ còn là một writer. Mỗi người đều có **trang web/blog**, **sách được xuất bản**, rất có tiếng tăm trong cộng đồng developer. Nhờ khả năng viết, những ý tưởng/dóng góp của họ được cộng đồng tiếp thu, cải tiến và trân trọng. Mình cũng viết blog để chia sẻ kiến thức, ý tưởng, tiếc là chưa tiếng tăm chưa được bao nhiêu.

Dưới đây là vài cái tên (theo mình) đã góp phần làm thay đổi bộ mặt của ngành Software Engineering chúng ta.



1. Dennis Ritchie

Ông là **cha đẻ của ngôn ngữ C** – ngôn ngữ mà hầu như developer nào cũng phải học khi “Nhập môn lập trình”. Ông còn góp phần vào phát triển hệ điều hành Unix, tiền thân của Linux và OS X. Ông mất ngày 12/10/2011, hưởng thọ 70 tuổi. Đám ma không kèn không trống, chẳng bù với Steve Jobs.

Sách đã xuất bản: Unix Programmer's Manual; The C Programming Language



2. [Uncle Bob](#)

Với hơn 35 năm tuổi nghề, ông được giới Software Engineer gọi bằng cái tên thân thương là “Chú Bob”. Ông là một trong những tác giả của [Agile Manifestor](#). Với lối viết vui nhộn, hài hước, ông viết về [Testing](#), [Software Design](#), [Design Pattern](#) rất sâu sắc vào dễ hiểu. Hai cuốn sách [Clean Code](#) và Agile Software Development là sách gối đầu của nhiều thế hệ developer.

Sách đã xuất bản: Clean Code: A Handbook of Agile Software Craftsmanship; The Clean Coder: A

Code of Conduct for Professional Programmers; Agile Software Development, Principles, Patterns, and Practices; ...



3. [Martin Fowler](#)

Ông là một lập trình viên, kiêm tác giả và diễn giả. Ông cũng là một trong những người “có cỗ phần” trong [Agile Manifestor](#). Những bài viết [blog](#) của ông về Agile, Refactoring, Inversion of Control, [Continuous Integration](#) được đánh giá là “tài liệu tham khảo kinh điển”.

Những sách ông đã xuất bản cũng gây ra tiếng vang lớn trong giới Software Engineer.

Cuốn [Refactoring](#) giúp mọi người nhận ra **tầm quan trọng của việc refactor code**. Bản thân mình từng viết 1 bài [tổng quát](#) về [NoSQL](#) nhờ đọc qua

cuốn NoSQL Distilled của ông.

Sách đã xuất bản: Refactoring: Improving the Design of Existing Code; Patterns of Enterprise Application Architecture; [NoSQL Distilled](#); ...



4. Joel on Software

Mình đã từng giới thiệu về ông ở bài trước “[Các blog IT đáng đọc](#)“. Ông nổi tiếng cũng nhờ các bài viết trên blog của mình. Mỗi bài viết đều được chăm chút kĩ lưỡng, khá hài hước, sâu sắc và bổ ích. Ông cũng là những người đầu tiên đưa ra lời khuyên: “Lập trình viên nên luyện kĩ năng viết lách (writing) song song với kĩ năng viết code (coding)“.

Sách đã xuất bản: User Interface Design for

Programmers; Joel on Software; More Joel on Software; *Smart and Gets Things Done*; ...



5. David Hansson

Anh (Sinh năm 79, vẫn còn trẻ) là **cha đẻ của Ruby on Rail** – một framework từng gây bao nhiêu sóng gió trong cộng đồng developer. Các framework ra đời sau như ASP.NET MVC cũng chịu ảnh hưởng từ RoR. Sau khi sáng lập công ty 37signals, anh mở thêm blog [signal VS noise](#), nơi chia sẻ kinh nghiệm về UI/UX và lập trình Ruby.

Mình từng đọc hai cuốn do anh viết: Getting Real, Rework, ... Getting Real (Ebook được 37signals phát [miễn phí](#)) không phải sách về lập trình, mà là sách về “cách làm một phần mềm tốt, được nhiều người dùng”. Cuốn Rework là tổng hợp những kinh nghiệm, kiến thức khi bạn muốn khởi nghiệp với

ngành IT. Cả 2 cuốn đều hay, các bạn nên tìm đọc.

Sách đã xuất bản: Agile Web Development with Rails; Getting Real; Rework; Remote: Office not Required; ...

Bạn nào muốn đóng góp thêm về các “Cây đa cây đề” trong ngành mình thì cứ thoải mái đóng góp trong phần comment nhé.

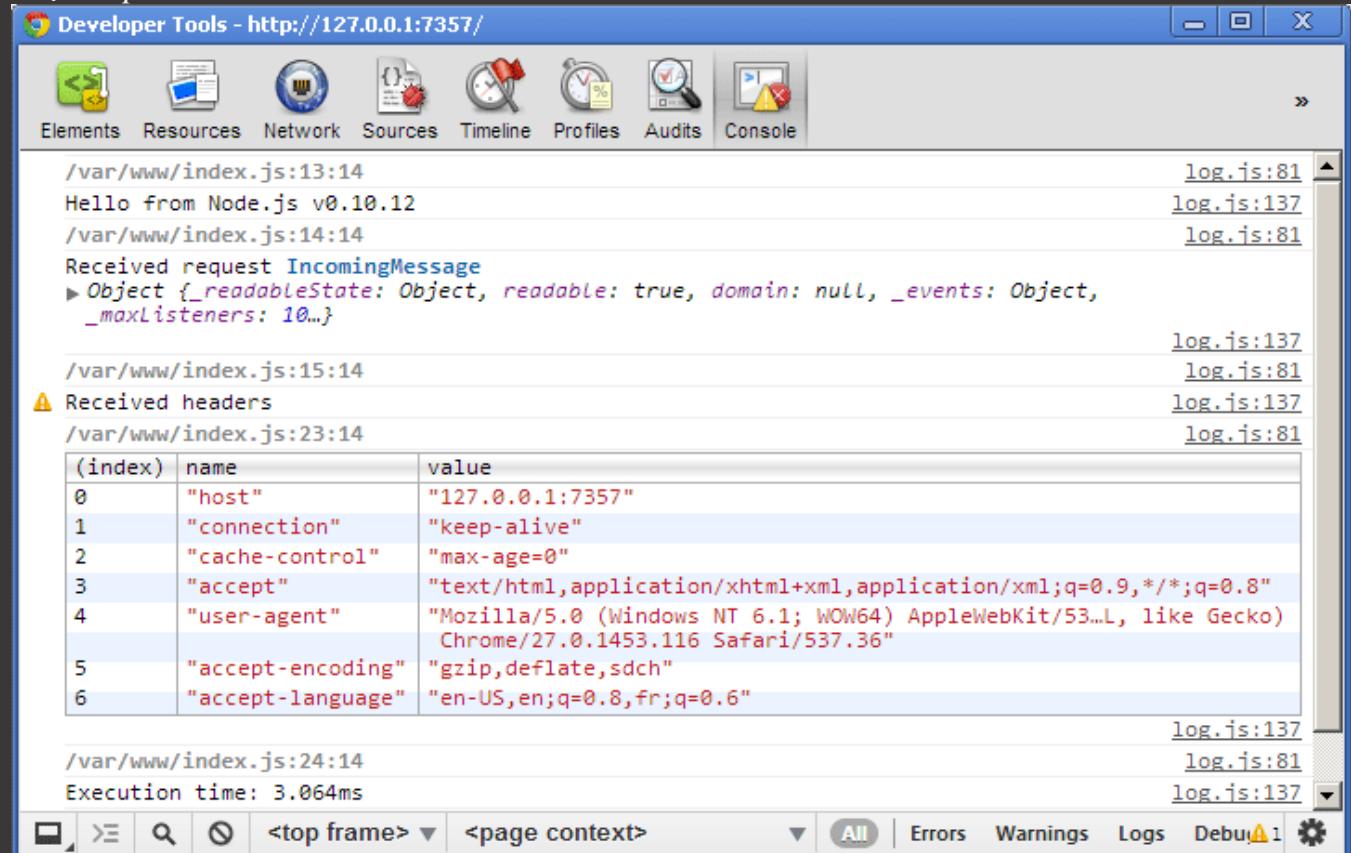
Sự “bá đạo” của Chrome Developer Tools – Phần 1

Posted on 08/10/2015 by Phạm Huy Hoàng

Với các bạn web developer, trình duyệt web là thứ chúng ta tiếp xúc nhiều nhất chỉ sau IDE. Chrome là một trình duyệt web được giới lập trình viên ưa chuộng vì nhanh, tiện lợi, và có bộ Developer Tools vô cùng mạnh mẽ. Bài viết này sẽ giới thiệu một số tính năng của bộ Developer Tools này.

Các bạn nào muốn theo nghiệp web developer nên đọc kĩ bài này nhé, vì bạn sẽ áp dụng những kiến thức này trong suốt quá trình làm việc đây (Dù bạn có làm C#, Java, PHP, hẽ có đụng tới front-end thì đều phải dùng Chrome Developer Tools cả).

Để mở cửa sổ Developer Tools, ta ấn rõ hợp phím **Ctrl + Shift + J**. Nhiều bạn quen sử dụng Developer Tools để debug CSS, có thể nhấp chuột phải vào một element, chọn *Inspect Element*.



Giao diện có thể hơi khác qua các phiên bản Chrome, nhưng số lượng và chức năng của các Tab vẫn tương tự nhau.

1. Elements

Tab này **hiển thị HTML của các element** trong trang web. Trước khi có Developer Tools, ta phải sửa CSS, save lại rồi refresh lại page. Ngày nay, ta có thể sửa trực tiếp CSS của 1 element vào khung phía bên phải, xem kết quả ngay lập tức. (Đây là một chức năng khá hay có từ add-on Firebug của Firefox, được các web developer/web designer dành tặng vô số lời khen).

Ngoài ra, nếu phải thiết kế web responsive, ta cũng có thể click vào **icon mobile** để test trang web trên màn hình các device với độ phân giải khác nhau.

The screenshot shows the Chrome DevTools Elements tab. On the left, the DOM tree displays the structure of a web page, including various HTML elements like <div>, <h1>, , and <script>. On the right, the Styles panel shows the computed styles for selected elements, allowing for real-time CSS editing. The page content includes a header with 'The Awesomeness of Chrome's Developer Console', a summary section, and a large image thumbnail.

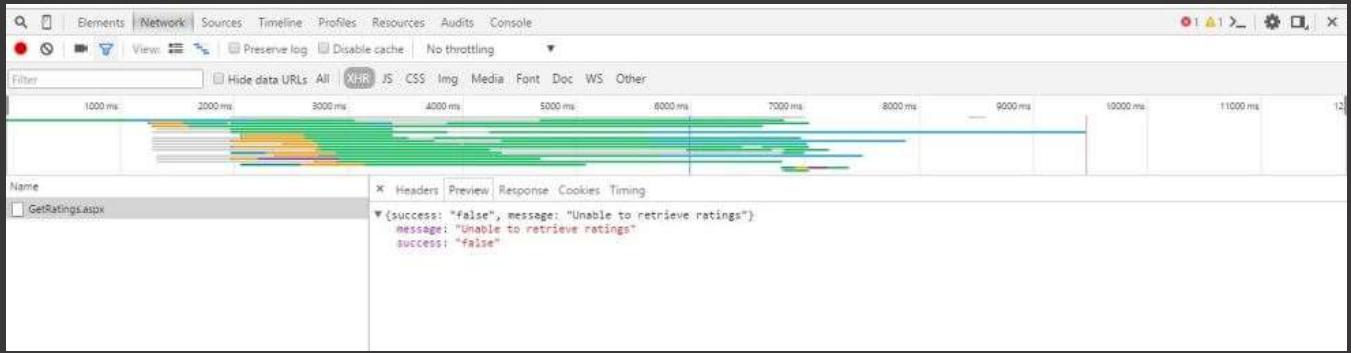
2. Networks

Ở tab này, ta có thể xem toàn bộ **những thứ được trình duyệt tải về** từ server: HTML, CSS, JS, hình ảnh, ... Mình thường dùng tab này để xem thời gian tải trang, nhằm *optimize* và *tăng tốc độ*.

The screenshot shows the Chrome DevTools Network tab. It displays a timeline of network requests, with each request represented by a colored bar indicating its duration. Below the timeline is a table of requests, showing details like Name, Method, Status, Type, Initiator, Size, and Time. The table lists various assets such as images, scripts, and stylesheets, along with their respective sizes and load times.

Name	Method	Status	Type	Initiator	Size	Time	Timeline - Start Time
translate_24dp.png	GET	204	ping	element_main[0]74	140 B	583 ms	
utm.gif?utmwww=5.6.8&utmms=4&utmhn=www.codeproject.com&utmcs=UTF-8&utm... client=te&alpha=true&hl=en&cb=_0lejhs5z	GET	200	gif	The-Awesomeness-of-C...	373 B	191 ms	
cleardot.gif	GET	200	script	element_main[0]292	806 B	187 ms	
translateelement.css	GET	304	gif	element_main[0]81	199 B	229 ms	
star-empty-lg.png	GET	304	stylesheet	element_main[0]31	0 B	177 ms	
data:image/png;base64,iVBORw0KGgoAAA...	GET	304	ping	jquery_min.js[18]	293 B	390 ms	
data:image/png;base64,iVBORw0KGgoAAA...	GET	(data)	ping	http://pubads.g.doublec...	0 B	0 ms	
data:image/png;base64,iVBORw0KGgoAAA...	GET	(data)	ping	page-fields[163]	0 B	0 ms	
datafont/woffbase...	GET	200	font	The-Awesomeness-of-C...	0 B	3 ms	

Các AJAX request cũng hiện trong Tabs này. Bạn có thể bấm vào từng request riêng lẻ để xem *thời gian request chạy, request đã gửi gì lên server, kết quả trả về từ server*. Đôi khi sử dụng AJAX nhưng code không chạy, mình phải sử dụng tab này để xem lỗi là gì, xảy ra ở client hay server.

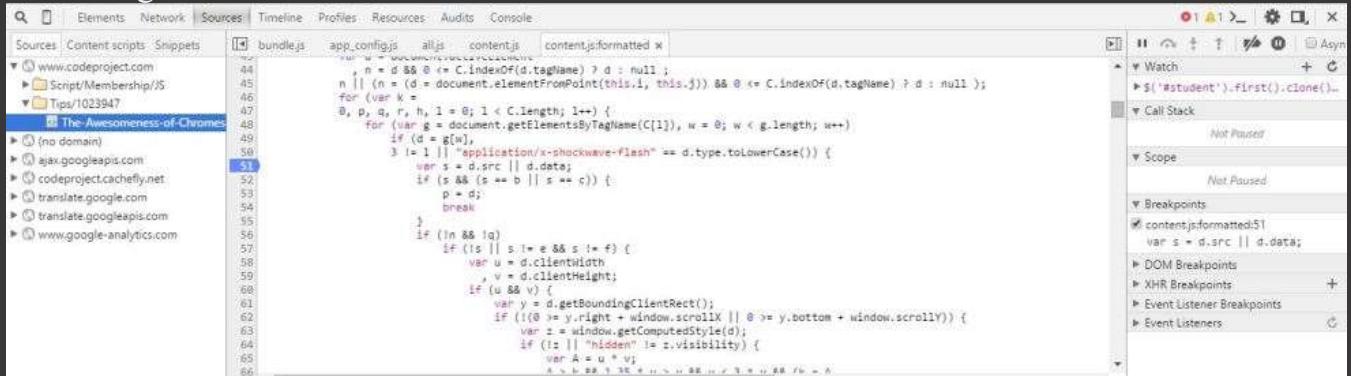


3. Sources

Tab này hiển thị những file javascript được trình duyệt load.

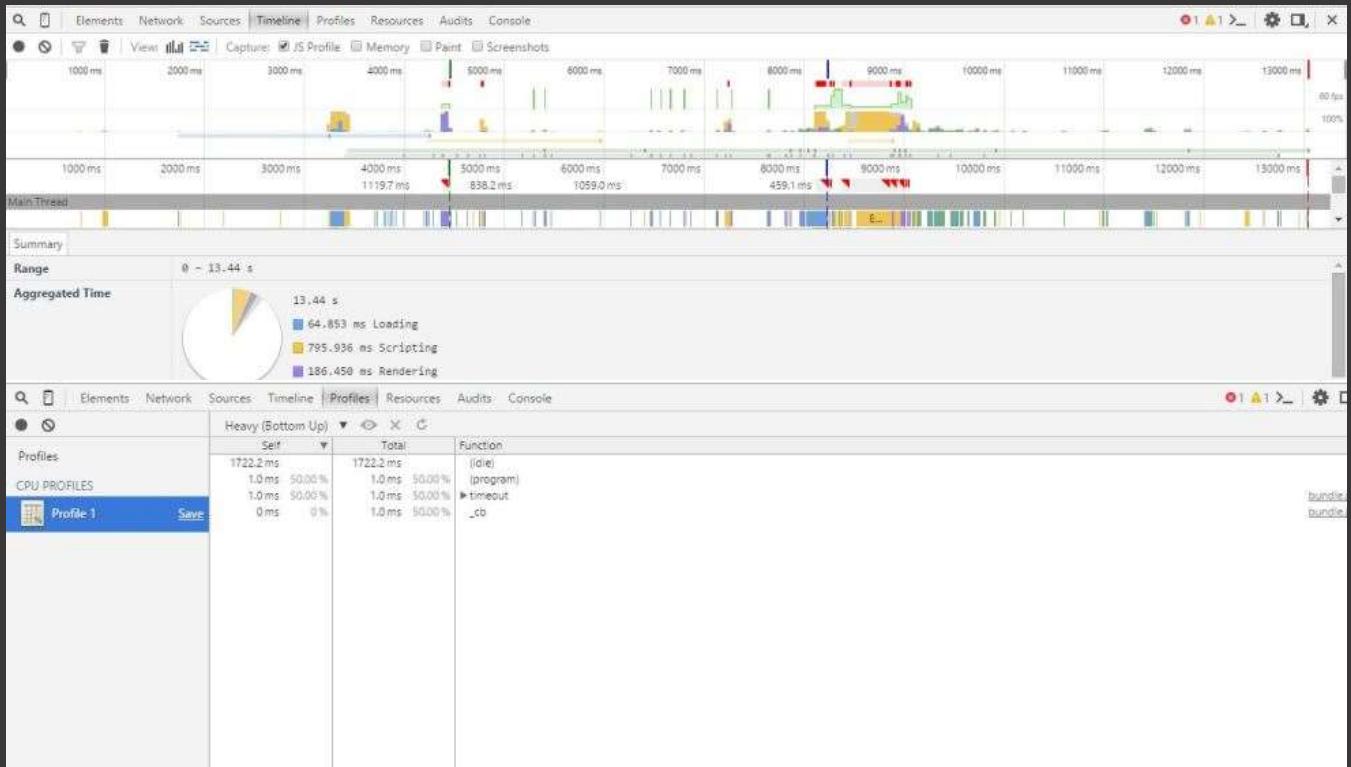
Ở tab này, ta có thể debug xem javascript chạy như thế nào. Với các ứng dụng sử dụng nhiều javascript hoặc dùng các library/framework như jQuery, AngularJS, ta có thể vào tab này, đặt breakpoint và debug như các IDE thông thường.

Có một mẹo nhỏ khi debug: Nếu các bạn code trong các trang [fiddle](#), hoặc làm việc với quá nhiều file, thư viện vài ngàn dòng, không thể đặt breakpoint bằng tay... hãy thêm dòng lệnh “debugger” vào sau dòng code cần debug. Bộ Debugger của Developer Tools sẽ tự xem dòng lệnh này là breakpoint, dừng ở câu “debugger” cho bạn debug



4. Timeline & Profile

Hai tab này mình ít khi dùng. Khi javascript chạy quá chậm, developer sẽ sử dụng 2 tabs này để tìm hiểu function nào chạy chậm, nhằm optimize lại code và tăng tốc cho trang.



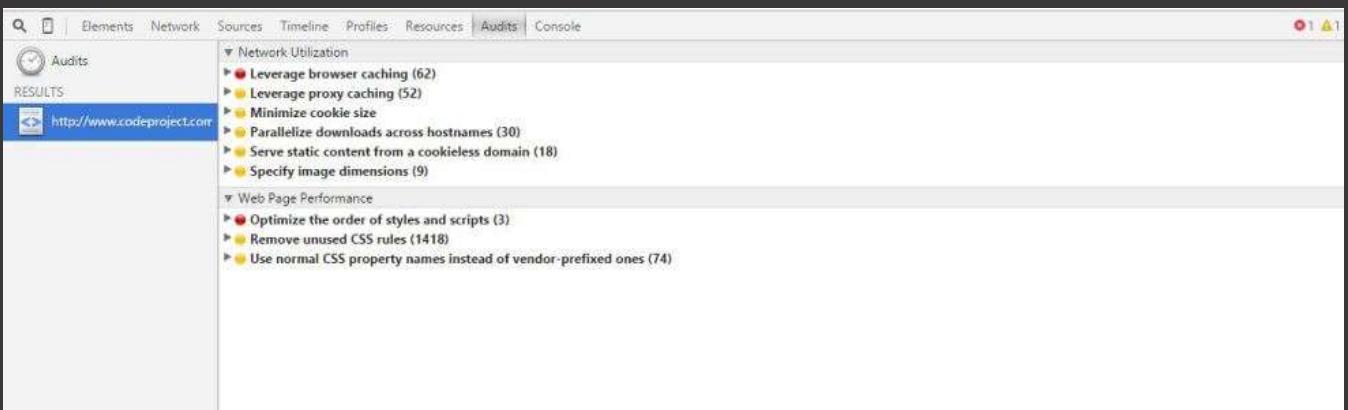
5. Resources

Tab này hiển thị **những thông tin được lưu trữ trong WebSQL, LocalStorage, SessionStorage, Cookies**. Khi sử dụng những công nghệ trên để lưu trữ dữ liệu người dùng, ta có thể vào tab này để kiểm tra

Frames	Name	Value	Domain	Path	Expires / Max-Age	Size	HTTP	Secure	First-Party
Web SQL	SessionGUID	ef1261c2-4cf7-4ebe-9581-c0e48565aca	www.codeproject.com	/	Session	47			
IndexedDB	_utma	40492976.412458933.1429535578.1433511473.1434279576.6	www.codeproject.com	/	2017-09-14..	59			
LocalStorage	_utmb	40492976.412458933.1429535578.1433511473.1434279576.6	www.codeproject.com	/	2017-09-14..	59			
http://www.codeproject.com	_utmc	40492976.5.10.1442328637	www.codeproject.com	/	2015-09-15..	30			
Session Storage	_utm	40492976	codeproject.com	/	Session	14			
http://www.codeproject.com	_utmt	1	codeproject.com	/	2015-09-15..	7			
Cookies	_utmx	40492976.1434279576.6.utmcsr=googleutmccn=(organic)utmcmd=...	codeproject.com	/	2016-03-16..	99			
www.codeproject.com	_utmc	40492976.1434279576.6.utmcsr=googleutmccn=(organic)utmcmd=...	www.codeproject.com	/	2016-03-16..	99			
	vk	63dcce95c-3609-492c-80ac-d3f82d04c203	codeproject.com	/	2016-04-20..	38			

6. Audits

Tab này có chức năng **kiểm duyệt tốc độ của website**, đưa ra những lời khuyên để tăng tốc. Vào Tab này và bấm Run, Developer Tool sẽ tiến hành đo đạc, so sánh tốc độ load của Web Site, kiểm tra xem Website đã được tối ưu hay chưa, sau đó đưa ra những cách tăng tốc. Một chức năng khá hay và thú vị.



7. Console

Những lỗi liên quan tới javascript (không load được, thiếu mở đóng ngoặc, chấm phẩy, ...) sẽ hiển thị trong tab này. Ta có thể nhập trực tiếp javascript vào tab này để chạy.

The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. The page being analyzed is 'Trang trên www.codeproject.com cho biết: Hello js'. The console output is as follows:

```
<top frame>
var obj = {name:'hoang',age:20,isBoy:true};
Object {name: "hoang", age: 20, isBoy: true}
alert('Hello js');
```

Nhờ khả năng chạy trực tiếp javascript, ta có thể làm rất nhiều trò “thú vị” với tab Console. Các bạn đón xem [phần 2](#) để biết nhé.

Bài viết tham khảo: <http://www.codeproject.com/Tips/1023947/The-Awesomeness-of-Chromes-Developer-Console>

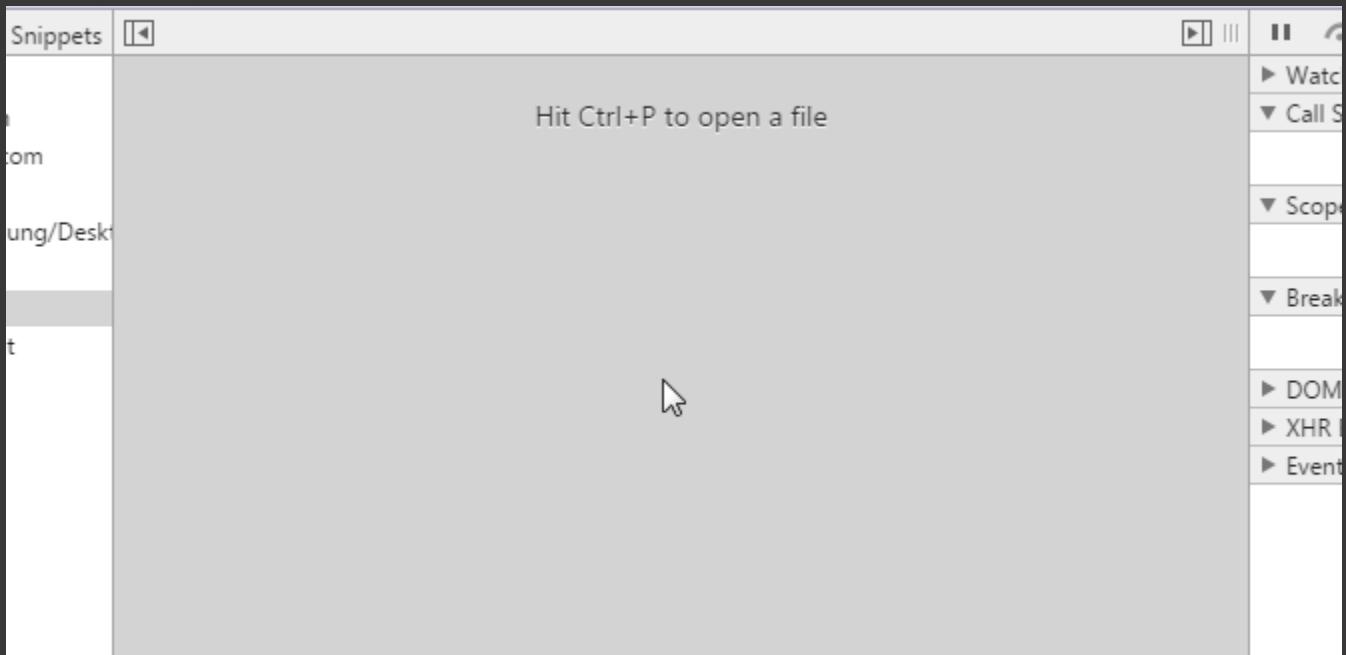
Sự “bá đạo” của Chrome Developer Tools – Phần 2

Posted on 13/10/2015 by Phạm Huy Hoàng

Ở [phần 1](#), mình đã giới thiệu với các bạn về chức năng cơ bản của các tab trong Chrome Developer Tools. Ở phần này, mình sẽ chia sẻ một số mánh khốe, chiêu trò hữu ích mà các bạn có thể áp dụng. Những chiêu này sẽ rất có ích khi code, debug, hoặc để lòe cấp trên và thành viên mới.

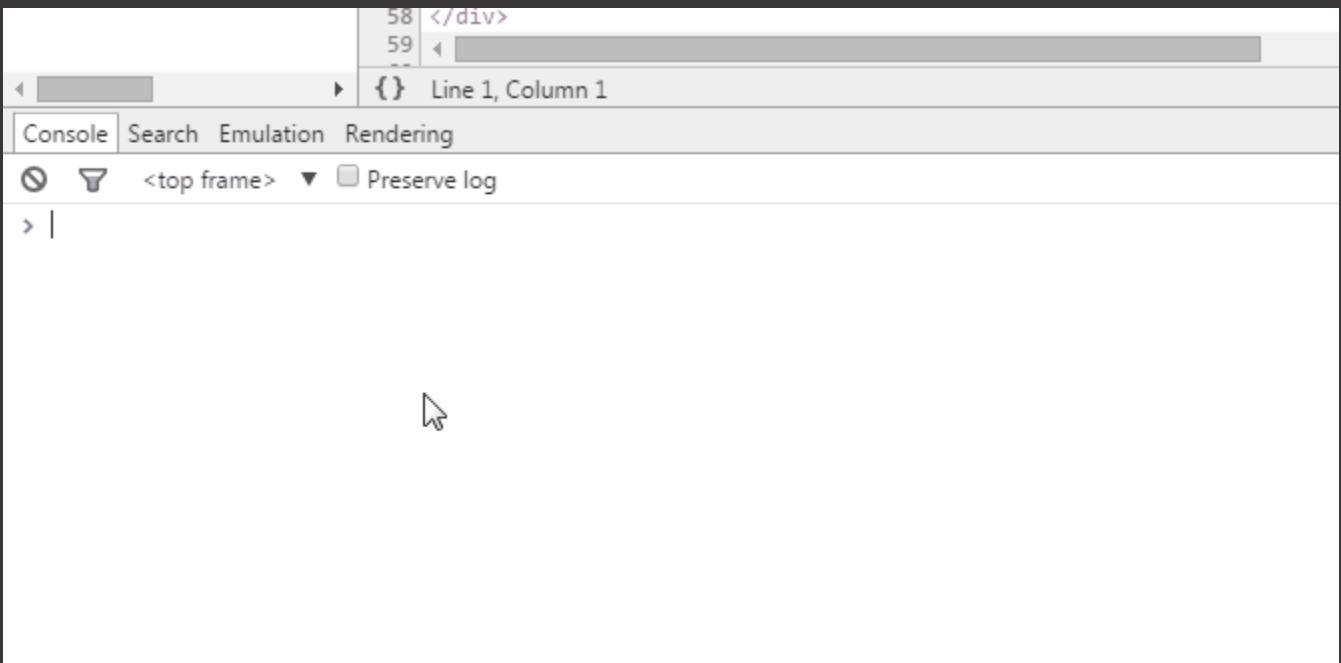
1. Mở nhanh file javascript

Với những dự án lớn, có hơn chục file css, js,... việc tìm mở file sẽ rất khó khăn. Chỉ cần mở Developer Tools, bấm tổ hợp Ctrl + P, những file cần tìm sẽ hiện ra đầy đủ, bạn chỉ việc chọn và ấn Enter.



2. Tìm kiếm trong source code

Thông thường, để tìm kiếm trong file html, css, js, ta thường dùng chức năng Search trong IDE. Developer Tool cũng có chức năng tương tự, kích hoạt bằng cách ấn Ctrl + Shift + F.



3. Select các element trong HTML

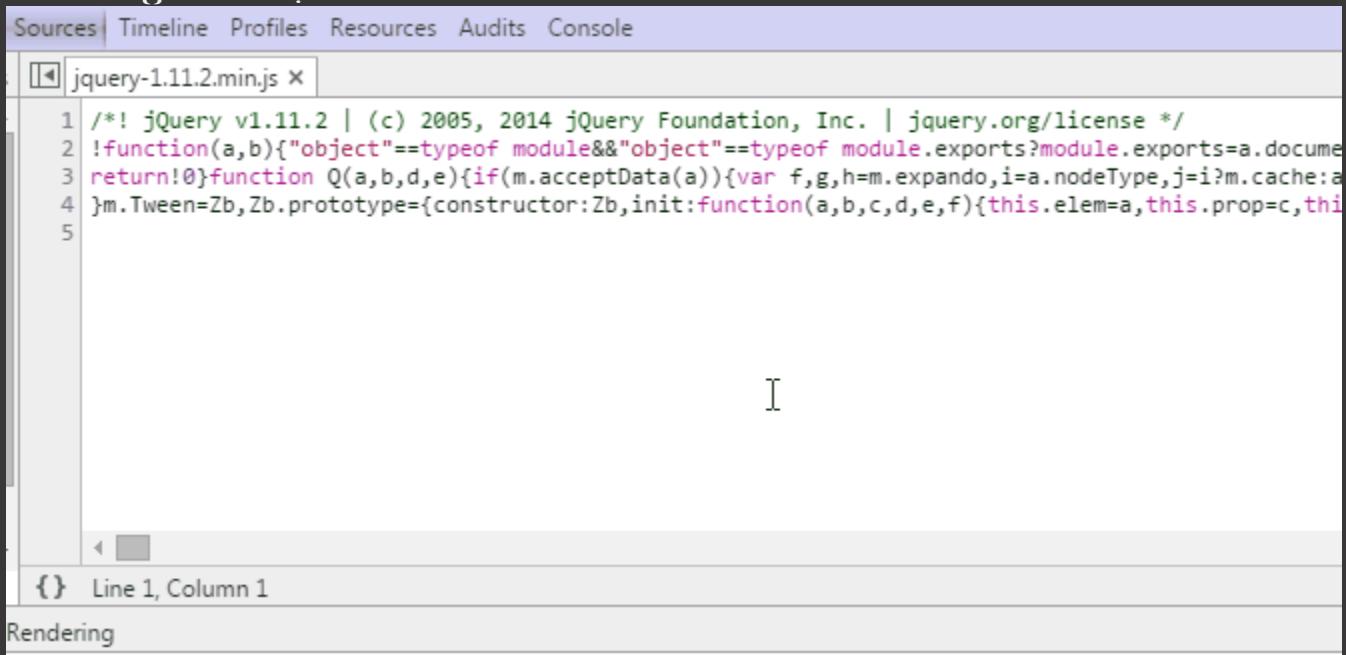
Để chọn 1 element trong HTML, ta thường hay sử dụng jQuery. Cửa sổ Console của Developer Tool cũng có 1 bộ api để ta select các element.

- **\$()**: Viết tắt cho **document.querySelector()**. bạn có thể dùng cú pháp css để select element tương tự jquery. Kết quả trả về là element đầu tiên tìm được.
- **\$\$()**: Viết tắt cho **document.querySelectorAll()**. Tương tự như **\$()**, nhưng trả về toàn bộ các element.
- **\$0-\$4**: Các element đã được select gần đây nhất. \$0 là element cuối cùng được select.



4. Chuyển code xấu thành code đẹp

Nhiều khi chúng ta viết code ấu, tất cả để chung một dòng, rất khó đọc. Hoặc ta sử dụng thư viện với code đã minify, rất khó đọc hay debug. Rất đơn giản, chỉ cần nhấp vào icon {} để “tút lại nhan sắc” cho code.



5. Chuyển đổi trạng thái của các element (hover, active)

Khi dùng Developer Tools kiểm tra css của các element, ta phải rê chuột lên element, hoặc click vào element đó, đợi khi khá rách rồi. Tab Element có một khung cho phép chúng ta set trạng thái của các element này luôn.

The screenshot shows the Chrome DevTools Styles tab. At the top, there are tabs: Styles, Computed, Event Listeners, DOM Breakpoints, and Properties. The Styles tab is selected. Below the tabs, there are two CSS rule sections:

```
.player-control div {  
    -webkit flex: 1;  
    flex: 1;  
    background-color: #fff;  
    text-align: center;  
    line-height: 60px;  
    height: 100%;  
    cursor: pointer;  
}  
  
* {  
    box-sizing: border-box;  
    margin: 0;  
    padding: 0;  
    -webkit-user-select: none;  
    -ms-user-select: none;  
    user-select: none;  
}
```

On the right side of each rule, the file name is listed: styles.css:141 and styles.css:7 respectively. There are also '+' icons to add new rules.

6. Chính sửa nhanh nội dung các element

Khi muốn chỉnh sửa nội dung (text) của các đoạn văn, các đề mục trong trang web, ta phải bật Developer Tool lên, chọn element cần chỉnh sửa, sau đó sửa HTML. Chỉ cần vào cửa sổ console, đánh dòng lệnh **document.body.contentEditable=true**, bạn có thể thoải mái sửa text của trang web một cách dễ dàng.

The screenshot shows the Chrome DevTools Console tab. At the top, there are tabs: Elements, Network, Sources, Timeline, Profiles, Resources, Audits, and Console. The Console tab is selected. In the main area, the command **document.body.contentEditable=true** is entered and executed. The output shows:

```
Failed to load resource: net::ERR_BLOCKED_BY_CLIENT  
⚠ 'webkitRequestAnimationFrame' is vendor-specific. Please use the standard 'requestAnimationFrame' instead.  
> document.body.contentEditable=true  
< true
```

7. Tìm những event được bind vào một element

Khi debug, mình luôn đau đầu vì không biết những hàm nào sẽ được gọi khi click vào một button, hover qua một div. Với Developer Tool, ta có thể gõ hàm **getEventListeners(element)** vào cửa sổ console, hoặc tìm những event được

bind vào element đó trong tab Elements.

The screenshot shows the Chrome Developer Tools interface with the 'Elements' tab selected. In the left panel, the DOM tree is displayed, showing a modal dialog with the class 'modal-dialog'. In the right panel, the 'Event Listeners' section is expanded, showing various event types (DOMContentLoaded, blur, click, focus, mousemove, mouseup) and their corresponding listeners. A specific listener for the 'click' event on the modal dialog is highlighted, with its code being executed in the console below:

```
> getEventListeners($('#firstName'))  
< ▾ Object {click: Array[1], focus: Array[1], keyup: Array[1]} ⓘ  
  ► click: Array[1]  
  ► focus: Array[1] // Lists all the events binded to #firstName  
  ► keyup: Array[1]  
> getEventListeners($('#firstName')).click[0].listener  
< function clickEvent(){  
    alert('Click Event');  
}  
                                         // Display the Listener binded to the click event of #firstName
```

The bottom status bar shows the current context: html#mac.js.consumer.chrome body#grid div.modal-dialog-bg

8. In object ra console nhờ log và table

Chắc 90% các bạn web developer đều biết dùng **console.log()** để in ra một object trong cửa sổ console, giúp việc debug được dễ dàng. Chrome Developer Tool còn có một hàm rất thú vị nhưng ít người biết, đó là hàm **console.table()**. Hàm này có thể in 1 mảng các object ra dưới dạng 1 bảng, rất trực quan và dễ nhìn.

Elements Network Sources Timeline Profiles Resources Audits Console Clockwork

< top frame ▼ Preserve log

Filter Regex All Errors Warnings Info Logs Debug □ Hide network messages

Console was cleared VM1140:2

undefined

(index)	firstName	lastName	age	family-table:13
mother	"Susan"	"Doyle"	32	
father	"John"	"Doyle"	33	
daughter	"Lily"	"Doyle"	5	
son	"Mike"	"Doyle"	8	

undefined family-table:1

▶ |


```
> var myArray=[{a:1,b:2,c:3},{a:1,b:2,c:3,d:4},{k:11,f:22},{a:1,b:2,c:3}]
< undefined
> myArray
< [▼ Object { a: 1, b: 2, c: 3 }, ▶ Object { f: 22, k: 11 }, ▶ Object { a: 1, b: 2, c: 3 } ] // Represented as an array of Objects
  a: 1
  b: 2
  c: 3
  ▶ __proto__: Object
> console.table(myArray)
VM895:2
```

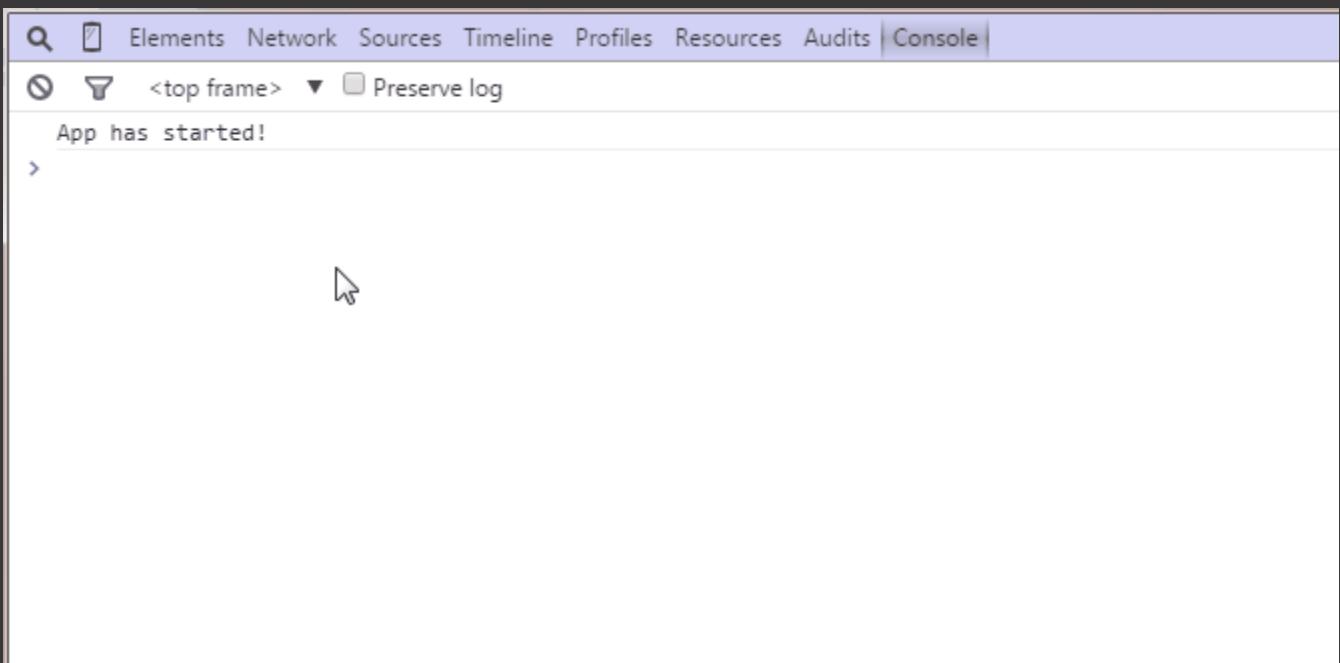
(index)	a	b	c	d	k	f
0	1	2	3			
1	1	2	3	4		
2					11	
3	1	2	3			22

< undefined // Represented in a table

▶

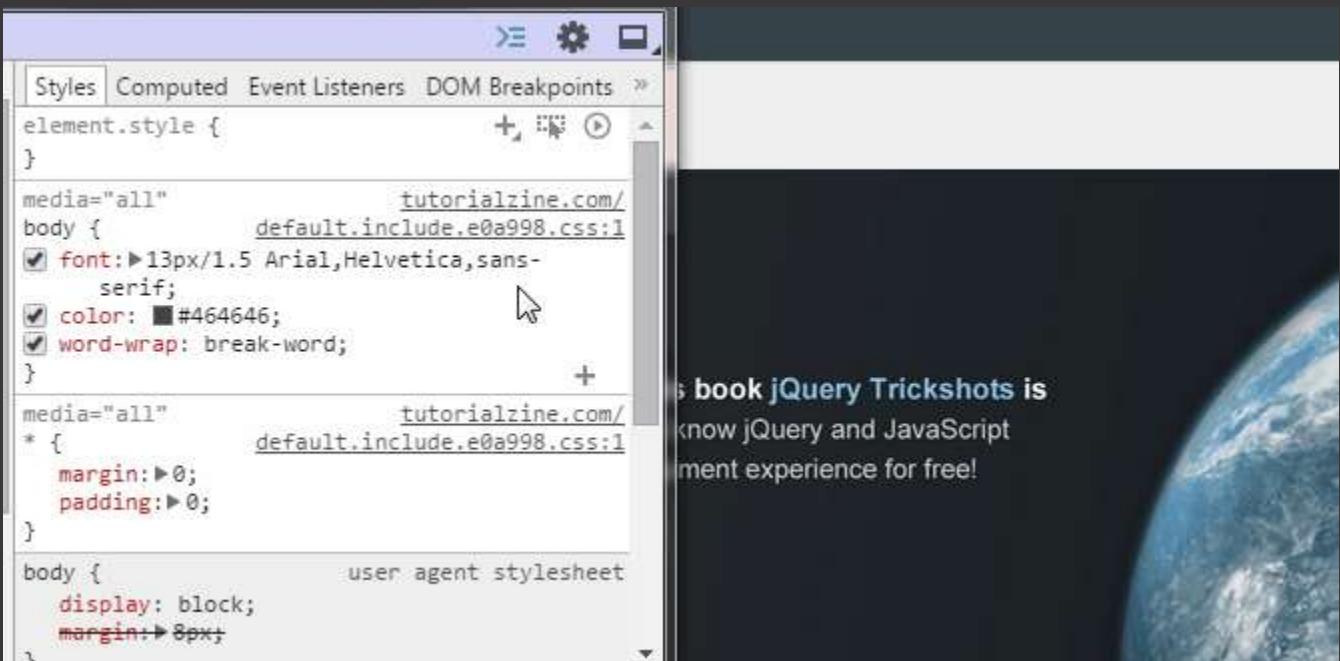
9. Lưu trữ console log, không mất log khi refresh lại trang

Một khi refresh lại trang web, cửa sổ Console sẽ bị xóa trống tron. Để nội dung cửa sổ Console vẫn giữ nguyên khi refresh lại trang, ta chỉ cần tick vào ô “Preserve Log”.



10. Lấy màu của các element khác trên trang web

Tab Element của Developer Tool có một công cụ để lấy màu vô cùng tiện lợi. Hãy click vào css nào liên quan tới màu sắc (color, background-color), một bảng màu sẽ hiện ra, cùng với một cây bút lấy màu cho phép bạn “chôm” màu từ element khác.



Bài viết được tham khảo từ nhiều nguồn. Các bạn có thể chia sẻ những kinh nghiệm làm việc với Developer Tools ở đây nhé.

Đôi dòng thông báo và cáo lỗi, kê lể cuộc sống ở UK

Posted on 01/10/2015 by Phạm Huy Hoàng

Nhu đã viết trong bài [chia tay ASWIG](#), mình đã nghỉ việc và qua UK học Master. Mình qua đến UK vào tối 25 – sáng 26. Do mình ở kí túc xá trường nên mọi chuyện cũng tạm ổn, mấy bạn tình nguyện viên hướng dẫn cũng khá thân thiện.

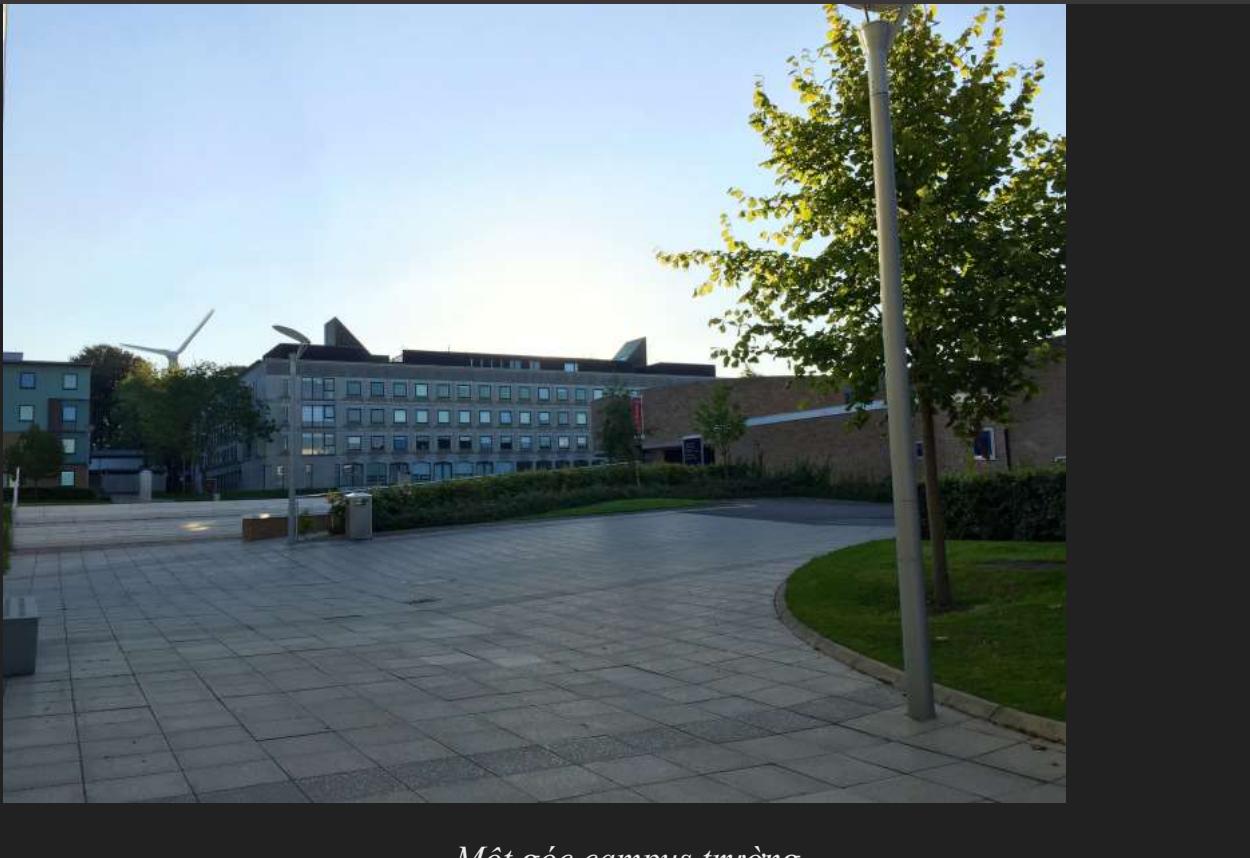


Kí túc xá

Mình ở chung kí túc xá với 2 bạn nữ Hàn Quốc (Chưa phẫu thuật), 2 bạn nam UK. Sinh viên trường từ khắp mọi nơi, đa phần là dân Trung Quốc, Ấn Độ, châu Âu, ... Đi vòng vòng mấy ngày mà không thấy đứa Việt Nam nào.

Campus của trường khá bụi, đi lòng vòng mệt đứt cả hơi, mỗi lần đi từ kí túc xá đến chỗ học phải đi bộ gần 10-15 phút. Nhiệt độ ngoài trời là 13 độ, đi đâu cũng phải khoác cái áo lạnh dày cui. Trường phủ sóng wifi 100% trong campus, khá là sướng,

tốc độ tải cũng được khoảng 7-12MB/s, tha hồ cắm phim (Nghe nói tải hàng copyright bị phạt tiền nên mình chưa dám làm).



Một góc campus trường

Trường cũng nằm khá xa trung tâm thành phố. Nếu muốn đi chơi phải chịu khó bắt xe buýt. Buýt ở đây giá cũng cao, lâu lâu mới có chuyến. May mà trường có event cho học sinh free trip để xuống thành phố mua đồ. Trong campus cũng có 4,5 cái siêu thị nhỏ nhở nhưng mua đồ ở ngoài vẫn tươi và rẻ hơn.



Một góc đường

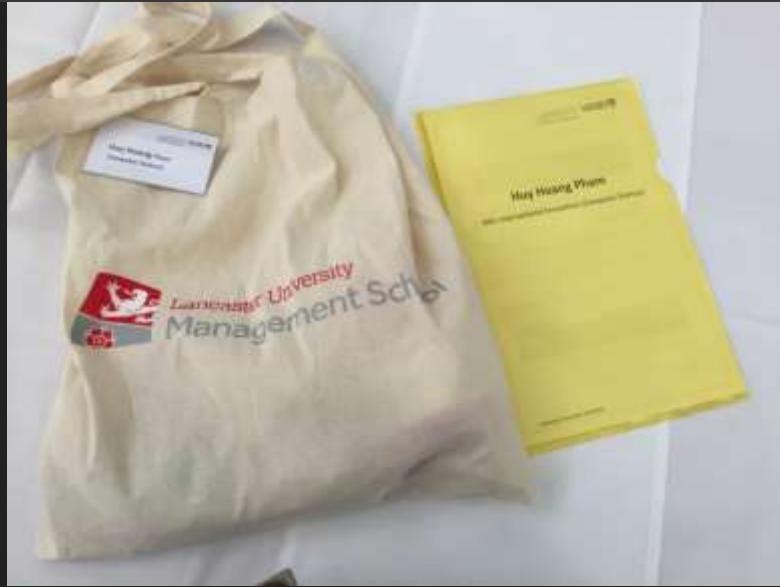
Bên này còn một cái ngô ngô nữa là: Máy đưa China gấp mình cứ tưởng China, xô 1 tràng tiếng Tàu, thấy mình ngơ ngơ mới hỏi tiếp “Are you Chinese?”. Mới đầu mình còn nhầm 2 bạn nữ HQ ở chung là Chinese mà, không trách tụi nó được. Gái bên này

cũng khá xinh, tiếc là mấy em mặt cute thì bươi lép, mấy em bươi khủng thì hơi béo phì, mặt cũng tròn vo :(.

Tuần đầu tiên (28/9 – 2/10) hầu như chỉ là đăng ký với lại giới thiệu chương trình thôi nên không có gì nhiều. Mới hôm đầu tiên vào mình đã được nghe quảng cáo trường thứ hạng cao thế này thế nọ, chuẩn này chuẩn kia. Làm quen được với 1 thằng Japan và 1 thằng Thailand, thằng Japan nói tiếng Anh cũng khá, không giống trong phim Nhật. *Ngành Computer Science của mình chỉ có 6 người, và không có bạn China nào (may quá).*



Bữa trưa free mừng SV mới



Quay lại thời sanh dzién



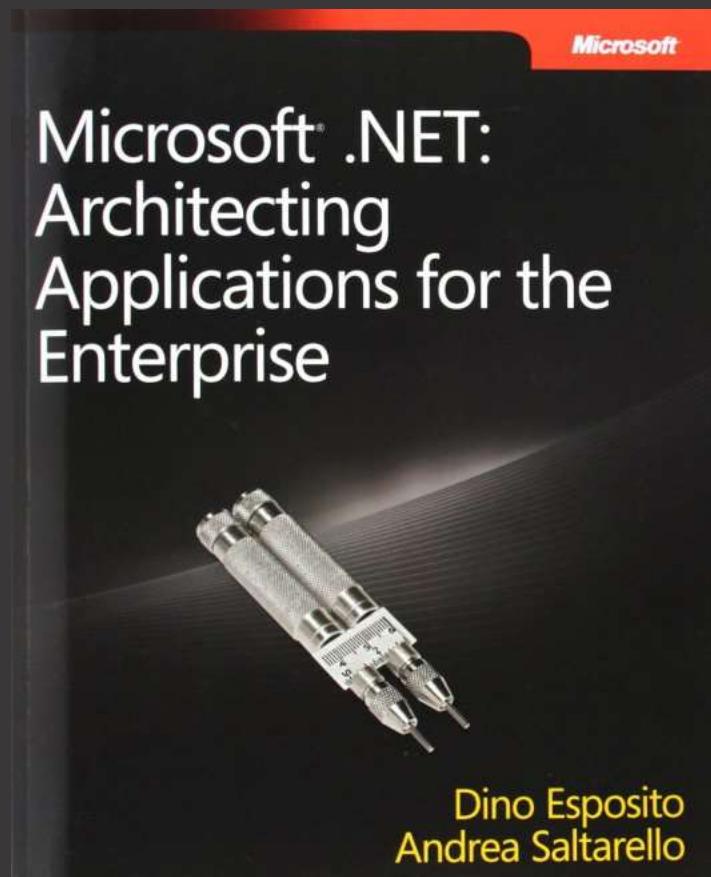
Đóng sách phải đọc trong học kì này (Còn 2 cuốn chưa mượn được)

Mấy hôm đầu vừa qua, mình phải đi mua sắm + tìm hiểu nhiều để quen dần với cuộc sống ở đây. Bên này sớm hơn so với VN 6 tiếng. Vì vậy, có thể mình không thể đăng bài đều đặn vào thứ 3 – thứ 5 như trước đây, hoặc lâu lâu bài đăng bị trễ (do lệch múi giờ). Mong các bạn đọc thông cảm và vẫn tiếp tục theo dõi blog như trước nhé. Cảm ơn sự ủng hộ của các bạn.

Review sách: Microsoft .NET – Architecting Applications for the Enterprise (1st Edition)

Posted on 06/10/2015 by Phạm Huy Hoàng

Mình có thói quen đọc sách cuối tuần, đủ các thể loại từ marketing, startup cho tới technical. Lâu rồi cũng chưa review cuốn sách nào nên thấy thiếu thiêu, dành review cuốn này vậy. Đây là một cuốn sách khá hay về thiết kế architecture cho các ứng dụng .NET.



Mặc dù nghe tên hầm hố nhưng sách không quá khó, các bạn từ junior, senior cho tới Software Architect đều có thể đọc hiểu cuốn này. Tuy nhiên, mình khuyên các bạn sinh viên hoặc mới ra trường đừng nên đọc. Đã làm được 1-2 năm, tiếp xúc với 1 số project lớn, bạn sẽ dễ hiểu những điều được viết trong sách hơn.

Cuốn sách này có 2 phiên bản: bản đầu tiên ra mắt năm 2008, bản 2nd Edition tái bản năm 2012. Hai phiên bản có **nội dung khác nhau đến 80%**, bài review này là của phiên bản đầu năm 2008.

Giới thiệu nội dung

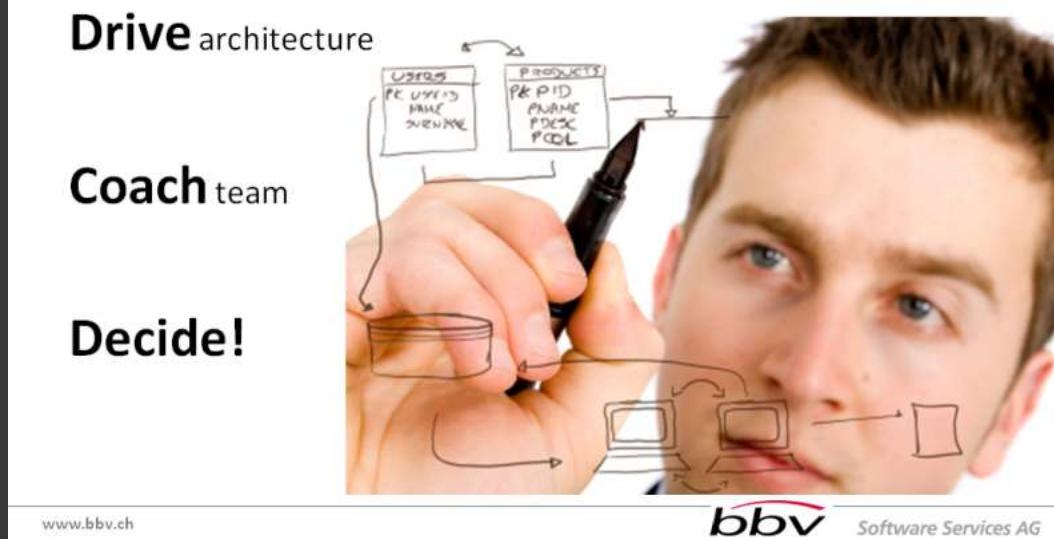
Tác giả cuốn sách là hai lão technical consultant khá cứng cựa, đã từng làm Software Architect cho vài chục dự án. Đi lên từ vị trí developer, những kinh nghiệm họ chia sẻ đều rất gần gũi với dân developer chúng ta. Bên cạnh những nội dung lý thuyết sách còn có những đoạn code mẫu nên khá dễ hiểu.

Ở chương đầu, tác giả định nghĩa lại về architecture: architecture là gì, thế nào là một architecture tốt,... Đây là những câu hỏi mà các bác developer cứng cựa còn khó trả lời. Kết luận rút ra: architecture là những thứ **khó thay đổi**, vai trò của người software architect là xác định đúng những thứ này vào giai đoạn đầu, xây dựng nền kiến trúc của hệ thống.

Chương sau, tác giả giới thiệu lại một số mô hình UML cơ bản (Chắc ở trường bạn nào cũng từng học về cái này trong môn: Cơ sở và nguyên lý thiết kế phần mềm). Các mô hình này đôi khi khá rối rắm và phức tạp, làm ta cảm thấy việc vẽ chúng khá là thừa thãi và mất thời gian. Tác giả đã đưa ra một kết luận rất hay: Mục đích của UML là để hỗ trợ giao tiếp giữa các bên trong dự án (dev, designer, tester, business analyst,...). UML giúp các bên hiểu nhau rõ hơn, chứ không phải để làm mất thời gian vẽ vời thiết kế của developer.

Cuối phần một, sách giới thiệu những qui tắc thiết kế cơ bản: Separation of Concern, các nguyên lý SOLID, các design pattern. Những kiến thức ở chương này khá có ích với developer ở mọi trình độ.

Duties of the Software Architect



Phần 2 là phần chính của cuốn sách, giới thiệu về 4 layer thường dùng trong các ứng dụng doanh nghiệp:

- Business Layer
- Service Layer
- Data Access Layer
- Presentation Layer

Sách chỉ rõ vài trò, nhiệm vụ của từng layer, cách lựa chọn công nghệ phù hợp, những design pattern nào nên áp dụng. Xuyên suốt những nội dung này, tác giả luôn nhắc lại một điều: **Không có design nào là hoàn hảo**, giải quyết được mọi vấn đề. Mọi design phải được dựa theo yêu cầu của business, người Software Architect phải nhận ra được các ưu/khuyết điểm của mỗi design để đưa ra lựa chọn phù hợp.

Nhận xét

Cuốn sách ra đời năm 2008, khi ASP. MVC, LINQ, Entity Framework còn ở giai đoạn sơ khởi, chưa hoàn thiện, do đó những lời khuyên về lựa chọn công nghệ không còn chính xác lắm. Tuy nhiên, những lời khuyên về kiến trúc, phân tích thiết kế và pattern vẫn rất có giá trị. Bạn sẽ vừa đọc vừa “*ồ ra thé!*” hoặc “*lẽ ra phải làm thế này mới đúng*”. Những kiến thức quý giá trong cuốn sách này sẽ giúp bạn: *hù doạ junior*,

bật lại senior và thể hiện trước mặt Software Architect. Nhớ khiêm tốn và đừng làm quá nhé.

Xem lẫn lý thuyết, sách luôn có nhiều đoạn code giải thích architecture, giải thích pattern. Có những đoạn đọc hoài không hiểu, chỉ cần giờ tới trang sau đọc code là “thông” ngay. Với những khái niệm phức tạp, tác giả còn kèm thêm nguồn tham khảo để ta tự tìm đọc thêm. Cuối sách, tác giả còn cung cấp **một solution gồm đủ 4 layer**, với 3 project riêng cho Web, Winform và WPF. Đây là những đoạn code thật, chạy được, **chuẩn hơn** các project code mẫu đầy rẫy trên mạng nhé.



Kết luận

Nếu chưa tin vào những lời quảng bá của mình, bạn hãy xem những [review](#) của cuốn sách trên amazon. Sách khá dày (Gần 400 trang) nên sẽ tốn kha khá thời gian đọc và thẩm đính nhé.

Như đã nhắc ở đầu bài, cuốn sách còn bản 2nd Edition năm 2012 nữa. Mình cũng đã đọc thử bản này. Bản 2012 tập trung giới thiệu **Domain Driven Design** (DDD) và **Command Query Responsibility Segregation** (CQRS). DDD là một hướng tiếp cận mới trong việc lấy requirement, viết user story, thiết kế hệ thống. [CQRS](#) là một pattern mới, tách rời việc đọc và ghi dữ liệu. Hai thứ này khá hay nhưng mình thấy các công ty Việt Nam ít áp dụng, do đó bản 2012 này không hữu dụng cho lắm so với bản 2008.

Muốn tìm thêm sách để nâng cao kỹ năng lập trình? Hãy thử đọc [Clean Code](#), [The Passionate Programmer](#), hoặc bài viết [này](#) nhé.

Series C# hay ho: EPPlus – Thư viện Excel “bá đạo” – Phần 2

Posted on 26/11/2015 by Phạm Huy Hoàng

Tiếp theo [phần 1](#), thì phần này mình sẽ hướng dẫn các bạn thao tác với các công thức trong Excel, cũng như đọc nội dung từ file Excel.

Thêm formula vào các ô trong Excel

Tương tự như Excel, chúng ta có thể dễ dàng gọi các formula như sau :

```
1 // Thực hiện tính theo formula trong excel
2 // Hàm Sum
3 worksheet.Cells[listItems.Count + 3, 3].Value = "Total is :";
4 worksheet.Cells[listItems.Count + 3, 4].Formula = "SUM(D2:D" + (listItems.Count + 1)
5 // Hàm SumIf
6 worksheet.Cells[listItems.Count + 4, 3].Value = "Greater than 20050 :";
7 worksheet.Cells[listItems.Count + 4, 4].Formula = "SUMIF(D2:D" + (listItems.Count + 1)
8 // Tính theo %
9 worksheet.Cells[listItems.Count + 5, 3].Value = "Percentatge: ";
10 worksheet.Cells[listItems.Count + 5, 4].Style.Numberformat.Format = "0.00%";
11 // Dòng này có nghĩa là ở column hiện tại lấy với địa chỉ (Row hiện tại - 1) / (Row hiện tại - 1)
12 worksheet.Cells[listItems.Count + 5, 4].FormulaR1C1 = "(R[-1]C/R[-2]C)";
```

Khá là dễ dàng phải không nào :D

Đọc nội dung từ file Excel

Tiếp sau đây mình sẽ hướng dẫn các bạn cách lấy data từ file excel và đưa lên web theo cách đơn giản nhất.

Bước 1. Viết hàm *ReadFromFile*

```
1 private DataTable ReadFromFile(string path, string sheetName)
2 {
3 // Khởi tạo data table
4 DataTable dt = new DataTable();
5 // Load file excel và các setting ban đầu
```

```

6 using (ExcelPackage package = new ExcelPackage(new FileInfo(path)))
7 {
8     if (package.Workbook.Worksheets.Count < 1) { // Log - Không có sheet nào tồn tại t
9         // Truyền vào name của Sheet để lấy ra sheet cần, nếu name = null thì lấy sheet đầu tiên
10        == sheetName)
11        ?? package.Workbook.Worksheets.FirstOrDefault();
12        // Đọc tất cả các header
13        foreach (var firstRowCell in workSheet.Cells[1, 1, 1, workSheet.Dimension.End.Column])
14        {
15            dt.Columns.Add(firstRowCell.Text);
16        }
17        // Đọc tất cả data bắt đầu từ row thứ 2
18        for (var rowNumber = 2; rowNumber <= workSheet.Dimension.End.Row; rowNumber++)
19        {
20            // Lấy 1 row trong excel để truy vấn
21            var row = workSheet.Cells[rowNumber, 1, rowNumber, workSheet.Dimension.End.Column];
22            // tạo 1 row trong data table
23            var newRow = dt.NewRow();
24            foreach (var cell in row)
25            {
26                newRow[cell.Start.Column - 1] = cell.Text;
27            }
28            dt.Rows.Add(newRow);
29        }
30    }
31    return dt;
32}

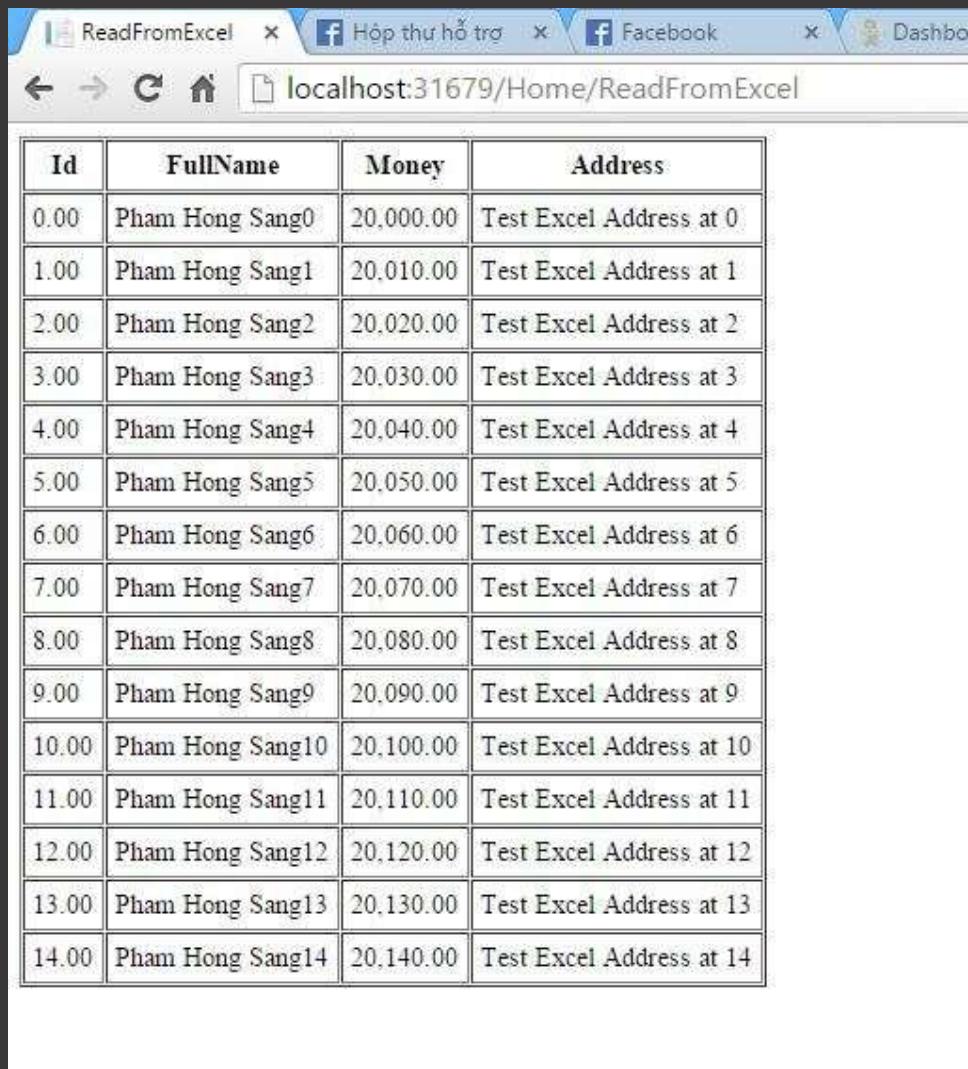
```

Lưu ý, trong ví dụ trên mình dùng *workSheet.Dimension*. Nó sẽ đọc tất cả các dòng và các cột đã được bind trong excel nên nếu bạn có dữ nhưng dòng trắng không cần thiết nó cũng sẽ đọc luôn. Nó sẽ lấy tới vị trí từ A1:D20 luôn nhé :D. Nếu các bạn không thích dùng cách này thì có thể pass 1 param vào

hàm *ReadFromFile* nhunumberOfColumn , chỉ lấy một số cột chứ không lấy toàn bộ nhé.

Bước 2. Thêm hàm *ReadFromFile* trong *HomeController*

```
1 [HttpGet]
2 public ActionResult ReadFromFile()
3 {
4     var data = ReadFromFile(@"D:\ExcelDemo.xlsx", "First Sheet");
5     return View(data);
6 }
7
8 @{
9     Layout = null;
10 }
11 <html>
12     <head>
13         <meta name="viewport" content="width=device-width" />
14         <title>ReadFromFile</title>
15     </head>
16     <body>
17         <div>
18             <table border="1" cellpadding="5">
19                 <thead>
20                     <tr>
21                         @foreach (System.Data.DataColumn col in Model.Columns)
22                         {
23                             <th>@col.Caption</th>
24                         }
25                     </tr>
26                 </thead>
27                 <tbody>
28                     @foreach (System.DataDataRow row in Model.Rows)
29                     {
```

The screenshot shows a web browser window with the URL localhost:31679/Home/ReadFromExcel. The page displays a table with four columns: Id, FullName, Money, and Address. The data consists of 15 rows, each containing a unique ID, a name starting with 'Pham Hong Sang' followed by a number, a monetary value, and a placeholder address.

Id	FullName	Money	Address
0.00	Pham Hong Sang0	20,000.00	Test Excel Address at 0
1.00	Pham Hong Sang1	20,010.00	Test Excel Address at 1
2.00	Pham Hong Sang2	20,020.00	Test Excel Address at 2
3.00	Pham Hong Sang3	20,030.00	Test Excel Address at 3
4.00	Pham Hong Sang4	20,040.00	Test Excel Address at 4
5.00	Pham Hong Sang5	20,050.00	Test Excel Address at 5
6.00	Pham Hong Sang6	20,060.00	Test Excel Address at 6
7.00	Pham Hong Sang7	20,070.00	Test Excel Address at 7
8.00	Pham Hong Sang8	20,080.00	Test Excel Address at 8
9.00	Pham Hong Sang9	20,090.00	Test Excel Address at 9
10.00	Pham Hong Sang10	20,100.00	Test Excel Address at 10
11.00	Pham Hong Sang11	20,110.00	Test Excel Address at 11
12.00	Pham Hong Sang12	20,120.00	Test Excel Address at 12
13.00	Pham Hong Sang13	20,130.00	Test Excel Address at 13
14.00	Pham Hong Sang14	20,140.00	Test Excel Address at 14

Qua 2 phần của bài viết, bạn đã có cái nhìn tổng quát về EPPlus và một số cách sử dụng cơ bản của thư viện này. Nếu có thắc mắc gì, các bạn cứ post vào mục comment nhé.

Series C# hay ho: EPPlus – Thư viện Excel “bá đạo” – Phần 1

Posted on 24/11/2015 by Phạm Huy Hoàng

Lang thang dạo qua các group trên facebook, mình thấy có nhiều bạn hỏi cách để xuất file Excel trên MVC, hoặc đọc nội dung từ file Excel trong C#. Bài viết này sẽ giới thiệu EPPlus, một thư viện C# khá “bá đạo”, có khả năng xử lý tuốt tuồn tuột những thứ liên quan đến Excel.

Bài này được viết bởi khách mời **Phạm Hồng Sang**, một bạn nam dễ thương cùng trường FPT và đồng nghiệp ASWIG với mình. Văn phong của Sang gãy gọn đơn giản chứ không lồng vòng hài hước như mình, bạn nào thấy giọng văn lạ thì đừng thắc mắc nhé.

Như cái tiêu đề ở trên thì trong series này mình sẽ hướng dẫn các bạn cách xử lý file excel trong C# một cách đơn giản nhất với thư viện EPPlus. Bạn có thể tải về tại [đây](#), hoặc cũng có thể tìm EPPlus trên Nuget và import vào. Toàn bộ source code của bài demo: <https://github.com/ToiDiCodeDaoSampleCode/EPPLus-Sample>

Đầu tiên, chúng ta sẽ tạo một class đơn giản, chứa thông tin các hàng trong Excel để thuận tiện cho việc demo.

```
1  public class TestItemClass
2  {
3      public int Id { get; set; }
4      public string FullName { get; set; }
5      public double Money { get; set; }
6      public string Address { get; set; }
7 }
```

Tiếp theo là tạo một list các item từ class mà ta đã tạo ở trên, các bạn cũng có thể sử dụng EF để lấy dữ liệu từ database cũng được. Để đơn giản và tiết kiệm thời gian, mình sẽ tạo một list data giả như sau:

```
1  private List<TestItemClass> CreateTestItems()
2  {
```

Các bước chuẩn bị đã xong, bây giờ chúng ta bắt đầu chế biến món ăn :D.

Tao File Excel

Chúng ta sẽ tạo file step by step như trong đoạn code dưới đây:

1. Truyền vào một stream hoặc Memory Stream để thao tác với file Excel.
 2. Thiết lập các properties cho file Excel.
 3. Cuối cùng là đổ data vào file excel thông qua hàm LoadFromCollection với params truyền vào là 3 tham số :
 - Collection: List các items
 - PrintHeader: True or False để in thêm cái header ra file excel
 - TableStyle: Chọn style cho table mà các bạn muốn :D
 4. Cuối cùng Save Sheet đó lại.

```

4         {
5             // Tạo author cho file Excel
6             excelPackage.Workbook.Properties.Author = "Hanker";
7             // Tạo title cho file Excel
8             excelPackage.Workbook.Properties.Title = "EPP test background";
9             // thêm tí comments vào làm màu
10            excelPackage.Workbook.Properties.Comments = "This is my fucking generated Co
11            // Add Sheet vào file Excel
12            excelPackage.Workbook.Worksheets.Add("First Sheet");
13            // Lấy Sheet bạn vừa mới tạo ra để thao tác
14            var workSheet = excelPackage.Workbook.Worksheets[1];
15            // Đỗ data vào Excel file
16            workSheet.Cells[1, 1].LoadFromCollection(list, true, TableStyles.Dark9);
17            // BindingFormatForExcel(workSheet, list);
18            excelPackage.Save();
19        }
20
21
22

```

Export file Excel

Sau khi đã tạo xong File Excel thì bây giờ chúng ta sẽ export file Excel đó ra và xem thành phẩm nhé. Bạn thêm hàm Export phía dưới vào *HomeController*, sau đó truy xuất tới đường link */Home/Export/* để xuất file excel ra.

```

1     [HttpGet]
2     public ActionResult Export()
3     {
4         // Gọi lại hàm để tạo file excel
5         var stream = CreateExcelFile();
6         // Tạo buffer memory stream để hứng file excel
7         var buffer = stream as MemoryStream;

```


A	B	C	D	E	F	G	H	I
1	Id	Full Name	Money	Address				
2	0	Pham Hor	20000	Test Excel Address at 0				
3	1	Pham Hor	20010	Test Excel Address at 1				
4	2	Pham Hor	20020	Test Excel Address at 2				
5	3	Pham Hor	20030	Test Excel Address at 3				
6	4	Pham Hor	20040	Test Excel Address at 4				
7	5	Pham Hor	20050	Test Excel Address at 5				
8	6	Pham Hor	20060	Test Excel Address at 6				
9	7	Pham Hor	20070	Test Excel Address at 7				
10	8	Pham Hor	20080	Test Excel Address at 8				
11	9	Pham Hor	20090	Test Excel Address at 9				
12	10	Pham Hor	20100	Test Excel Address at 10				
13	11	Pham Hor	20110	Test Excel Address at 11				
14	12	Pham Hor	20120	Test Excel Address at 12				
15	13	Pham Hor	20130	Test Excel Address at 13				
16	14	Pham Hor	20140	Test Excel Address at 14				
17								
18								
19								

Đây là thành phẩm, trông hơi cùi bắp và khá chuối phải không =))). Ở bước tiếp theo, mình sẽ hướng dẫn các bạn format file excel sao cho đẹp mắt hơn.

Tút lại nhan sắc cho file Excel

Dưới đây là đoạn code để format cho file Excel, các bạn nhớ uncomment đoạn `BindingFormatForExcel` trong hàm `CreateExcelFile` nhé.

```

1     private void BindingFormatForExcel(ExcelWorksheet worksheet, List<TestItemClass> list
2     {
3         // Set default width cho tất cả column
4         worksheet.DefaultColWidth = 10;
5         // Tự động xuống hàng khi text quá dài
6         worksheet.Cells.Style.WrapText = true;
7         // Tạo header
8         worksheet.Cells[1, 1].Value = "ID";
9         worksheet.Cells[1, 2].Value = "Full Name";
10        worksheet.Cells[1, 3].Value = "Address";

```

```
10     worksheet.Cells[1, 4].Value = "Money";
11
12     // Lấy range vào tạo format cho range đó ở đây là từ A1 tới D1
13     using (var range = worksheet.Cells["A1:D1"])
14     {
15         // Set PatternType
16         range.Style.Fill.PatternType = ExcelFillStyle.DarkGray;
17         // Set Màu cho Background
18         range.Style.Fill.BackgroundColor.SetColor(Color.Aqua);
19         // Canh giữa cho các text
20         range.Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;
21         // Set Font cho text trong Range hiện tại
22         range.Style.Font.SetFromFont(new Font("Arial", 10));
23         // Set Border
24         range.Style.Border.Bottom.Style = ExcelBorderStyle.Thick;
25         // Set màu ch Border
26         range.Style.Border.Bottom.Color.SetColor(Color.Blue);
27     }
28
29     // Đỗ dữ liệu từ list vào
30     for (int i = 0; i < listItems.Count; i++)
31     {
32         var item = listItems[i];
33         worksheet.Cells[i + 2, 1].Value = item.Id + 1;
34         worksheet.Cells[i + 2, 2].Value = item.FullName;
35         worksheet.Cells[i + 2, 3].Value = item.Address;
36         worksheet.Cells[i + 2, 4].Value = item.Money;
37         // Format lại color nếu nhu thỏa điều kiện
38         if (item.Money > 20050)
39         {
40             // Ở đây chúng ta sẽ format lại theo dạng fromRow,fromCol,toRow,toCol
```


66

67

68

Bây giờ thì bạn hãy build lại rồi Export file excel đó ra thử nhé :D. Bạn đã export thành công 1 file excel rồi đó.

A	B	C	D	E
1	1 Sang	Test Excel		
2	1 Sang0	Address at 0	\$20,000.00	
3	2 Sang1	Test Excel		
4	2 Sang1	Address at 1	\$20,010.00	
5	3 Sang2	Test Excel		
6	3 Sang2	Address at 2	\$20,020.00	
7	4 Sang3	Test Excel		
8	4 Sang3	Address at 3	\$20,030.00	
9	5 Sang4	Test Excel		
10	5 Sang4	Address at 4	\$20,040.00	
11	6 Sang5	Test Excel		
12	6 Sang5	Address at 5	\$20,050.00	
13	7 Sang6	Test Excel		
14	7 Sang6	Address at 6	\$20,060.00	
15	8 Sang7	Test Excel		
16	8 Sang7	Address at 7	\$20,070.00	
17	9 Sang8	Test Excel		
18	9 Sang8	Address at 8	\$20,080.00	
19	10 Sang9	Test Excel		
20	10 Sang9	Address at 9	\$20,090.00	
21	11 Sang10	Test Excel		
22	11 Sang10	Address at 10	\$20,100.00	
23	12 Sang11	Test Excel		
24	12 Sang11	Address at 11	\$20,110.00	
25	13 Sang12	Test Excel		
26	13 Sang12	Address at 12	\$20,120.00	
27	14 Sang13	Test Excel		
28	14 Sang13	Address at 13	\$20,130.00	
29	15 Sang14	Test Excel		
30	15 Sang14	Address at 14	\$20,140.00	
31		Total is:	\$301,050.00	
32		Greater than		
33		20050:	\$180,900.00	
34		Percentage:	60.09%	

Ở phần sau, mình sẽ hướng dẫn các bạn cách thao tác với các Formula trong Excel khi sử dụng EPPlus, cùng với cách lấy data trong file Excel (Read data from Excel file) để thao tác các business khác.

Link tham khảo : https://www.paragon-inc.com/resources/blogs-posts/easy_excel_interaction_pt5.

Mặt tối của ngành công nghiệp IT – Phần 2

Posted on 19/11/2015 by Phạm Huy Hoàng

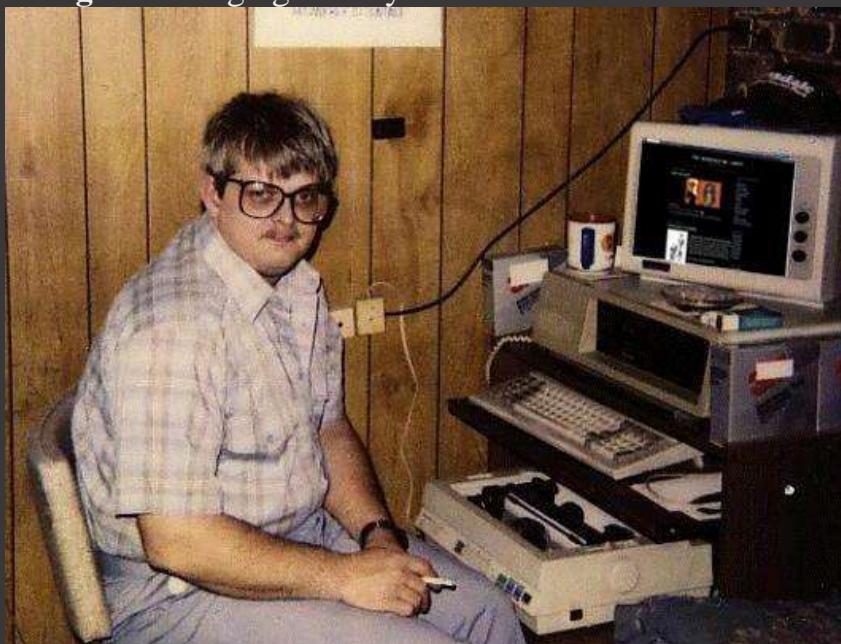
Nối tiếp [phần 1](#), bài viết này sẽ tiếp tục giới thiệu một số góc khuất ít người biết về ngành IT.

4. Giới technical không chưa đủ

Nhắc lại một câu nói từ [bài cũ](#):

Một điều khá may mắn trong ngành IT là: Vì đây là một ngành nặng về kỹ thuật. do đó bạn chỉ cần giỏi tập trung trau dồi technical cho giỏi. Chỉ cần technical giỏi, bạn sẽ được đồng nghiệp coi trọng, cấp trên tin tưởng giao phó trách nhiệm. Chỉ cần technical giỏi, con đường sự nghiệp của bạn sẽ rộng thênh thang, bạn sẽ nhanh chóng leo lên vị trí [senior, team leader, technical lead](#), ... Chỉ cần technical giỏi, lương bạn sẽ tăng vù vù, từ 500\$, 1000\$, 2000\$, các quảng cáo tuyển dụng toàn cầu người giỏi technical còn gì?

Nhiều bạn ngáo ngơ mới ra trường vẫn tưởng điều này là đúng, thấy mình hơi giỏi nên tỏ thái độ khi [phỏng vấn](#). Mỗi trường làm việc đòi hỏi nhiều thứ hơn cả khả năng code, đó là: khả năng giao tiếp, khả năng trình bày/[thuyết trình](#), kỹ năng [thương thảo](#), khả năng làm việc nhóm, ... Tới những cấp cao hơn như [team lead, quản lý](#), bạn còn phải biết cách quản lý, chia việc, cách thuyết phục, nói cho người khác nghe. Khả năng code là bắt buộc, nhưng chỉ code không là chưa đủ nếu bạn muốn **sống sót và thăng tiến** trong ngành này.



5. Cấp trên thường “đéo biết gì cả”

Đôi khi nói cấp trên “đéo biết gì” cũng không sai. Một sự thật buồn cười của ngành mình là: **Nếu giỏi technical, developer sẽ được đẩy lên làm quản lý.** Ở cấp quản lý, họ ít có thời gian code, mà phải lo chia việc, quản lý nhân sự, quản lý dự án,... toàn những việc trước giờ họ chẳng bao giờ làm. Ở cương vị mới, họ cũng phải học dần dần về quản lý, giống như chúng ta mới bắt đầu học code vậy thôi.

Các bạn lập trình viên thường có suy nghĩ rằng “Ông team leader/PM có code bao giờ đâu, công nghệ mới cũng *éo biết*. Điều này đúng, vì họ còn phải **lo nhiều việc khác**: nhân sự, tài nguyên, phỏng vấn ứng viên, hoặc đấu đá nội bộ công ty,... không có đủ thời gian để trau dồi technical. Đôi khi tranh cãi về một vấn đề, bạn chỉ thấy vấn đề technical, họ còn phải chú ý đến nhiều khía cạnh khác như: sắp xếp team, tiến độ dự án, trình độ thành viên, ...

Do vậy, **đừng giữ thái độ thù hằn, cay cú hay xem thường cấp trên.** Hãy cứ cố gắng thấu hiểu và học hỏi họ đi, biết đâu sẽ có lúc bạn “bị” đẩy lên làm quản lý như họ thôi. Gợi ý nhỏ: Nếu cấp trên thật sự “ngu” và bạn không chịu nổi, cứ thoái mái chuyển công ty/chuyển phòng ban khác thôi, ngành mình cũng dễ xin việc lắm.



6. Lập trình viên dần đang mất giá

Hôm trước, mình nhận được mail giới thiệu việc làm thêm của vlance – Một trang web cho freelancer Việt Nam. Nhìn giá cả các dịch vụ IT ở Việt Nam mà thấy thương: Một ứng dụng chỉ có giá 2 triệu, một website chỉ khoảng 1 triệu. Một phần do các công ty IT ở Việt Nam phá giá, làm mọi thứ từ web tới app với giá rẻ mạt. Một phần do các bạn developer Việt Nam đang bán rẻ chất xám, bán rẻ sức lao động của chính mình.

LÀM APP TỪ ĐIỂN SONG NGỮ VIỆT - TRUNG CHO HĐH ANDROID VÀ IOS

Ứng dụng di động | 2.000.000 VNĐ | Toàn Quốc

Hiện bên mình đang cần tìm một bạn developer có kinh nghiệm làm ứng dụng tra từ điển song ngữ Việt - Trung, về cơ bản tương tự ứng dụng <https://play.google.com/store/apps/details?id=com.thomvny.trungv...> Xem thêm

GỬI CHAO GIÁ

LÀM GAME LOẠN 12 SỨ QUẦN TRÊN NỀN IOS VÀ ANDROID

Ứng dụng di động | 2.000.000 VNĐ | Hà Nội

Làm game Loạn 12 sứ quân trên nền IOS và android. Minh muốn đặt hàng game Loạn 12 sứ quân trên nền IOS và android. Kiểu game như game kim cương, nhưng minh muốn xây dựng kiểu như game candy crush.... Xem thêm

GỬI CHAO GIÁ

TẠO LINK XEM PHIM TAM THỜI BẰNG PHP

Lập trình web | 1.000.000 VNĐ | Toàn Quốc

Mình muốn làm 1 trang xem phim online nhưng khi mình gắn link vào player để xem phim online. Ví dụ để file.mp4 lên host và để ở dir data thay vì xem link trực tiếp domain.com/data/file.mp4 nhưng muốn có 1... Xem thêm

GỬI CHAO GIÁ

[HTML, CSS, JS] CẦN TUYỂN PART TIME FREELANCER

Lập trình web | 1.500.000 VNĐ | Toàn Quốc

Hi, Hiện mình đang có nhu cầu tuyển dụng một Freelancer thành thạo: - HTML - CSS - JavaScript Mục đích là code Webpage Mobile responsive optimized đơn giản! Yêu cầu làm việc với mình mỗi b... Xem thêm

GỬI CHAO GIÁ

Ở đầu bài, mình từng nói rằng ngành lập trình viên **có mức lương kha khá**. Với các vị trí như senior, team leader, PM thì lương 1000-3000\$/tháng là chuyện bình thường. Tuy nhiên, mức lương ở các vị trí thấp hơn lại ... khá là thấp. Một phần là do nhiều bạn trẻ đỗ xô đi học công nghệ thông tin, học lập trình, khi ra trường lại không đủ trình độ, đành ngậm ngùi làm những công việc nhảm chán, mức lương thấp. Công nghệ hiện đại làm mọi người dễ tiếp cận việc lập trình hơn, nên người dùng có thể làm web, làm ứng dụng dễ dàng. Lập trình viên thì ngày càng “lười” hơn, “nhàn” hơn nhờ các framework. Trong tương lai, có thể ngành lập trình sẽ không còn hot như bây giờ, lập trình viên sẽ rót giá “thê thảm”.



Kết. Thέ giờ em phải sống nàm thao?

Nghe mình “hù” nay giờ, chắc bạn cũng hơi nghi ngại rồi phải không? Đừng lo, hãy áp dụng một số lời khuyên dưới đây của mình.

- Năm vững kiến thức nền tảng, học nhiều ngôn ngữ chứ đừng ôm khu khu 1ngôn ngữ. C++, C#, Java, ... có hay đến đâu cũng chỉ là công cụ. Quan trọng là bạn làm được gì với ngôn ngữ mình biết.
- Ngành mình có rất nhiều developer, nhưng có rất ít developer giỏi. Hãy tự trao dồi kiến thức và kĩ năng để tự nâng giá bản thân nhé, đã giỏi thì không lo thất nghiệp hay lương thấp.
- Xác định con đường mình muốn đi: Manager hay Technical, hoặc mở công ty riêng. Học thêm mấy thứ ngoài luồng như: quản trị, kinh tế, khởi nghiệp,... sau này sẽ có ích lắm đấy.
- Sóng khôn hơn và đừng ngáo ngơ nữa. Chịu khó đi xã giao, tạo dựng danh tiếng, quan hệ. Hãy đọc blog của một số developer nổi tiếng để học cách suy nghĩ, cách làm việc của họ.



Bạn nào muốn chia sẻ thêm về các mặt tối của ngành, hay có lời khuyên gì muốn đóng góp, cứ comment chia sẻ với mọi người nhé.

Mặt tối của ngành công nghiệp IT – Phần 1

Posted on 17/11/2015 by Phạm Huy Hoàng

Ngành lập trình kẽ ra cũng có khá nhiều cái sướng: Dễ xin việc, công việc thú vị, tiếp xúc nhiều cái mới, mức lương khá. Tuy vậy, nó có không ít mặt tối mà chỉ những người có thâm niên, tiếp xúc lâu với nghề mới trải nghiệm và nhận ra được. Bài viết này lấy cảm hứng từ course cùng tên trên pluralsight: [Technology Career Dark Side](#), nhằm giúp bạn đọc có cái nhìn khách quan hơn về ngành IT, cũng như tự rút ra cách “sống sót” cho bản thân mình.



1. Công nghệ liên tục thay đổi – Ai là người có lỗi

Trong ngành lập trình, [công nghệ là thứ liên tục thay đổi](#). Những công nghệ mới liên tục ra đời thay thế công nghệ cũ, làm kiến thức rất dễ lỗi thời vì. Do đó, người lập trình viên phải liên tục học hỏi, nếu không họ sẽ trở nên lạc hậu.

Nguyên nhân sâu xa đằng sau chuyện này chính là **Tiền**. Tại sao mỗi năm FIFA và PES đều ra bản mới? **Để bán lấy tiền**. Tại sao mỗi năm Iphone lại ra phiên bản mới? **Để hút máu người dùng, để kiếm tiền**. Đó cũng là lý do các hãng công nghệ liên tục đưa ra các sản phẩm/công nghệ mới để **bán lấy tiền**: [C# thay đổi từ 2.0 tới 5.0](#), Windows mỗi 2-3 năm lại ra bản mới, Visual Studio và SQL Server cũng tương tự.



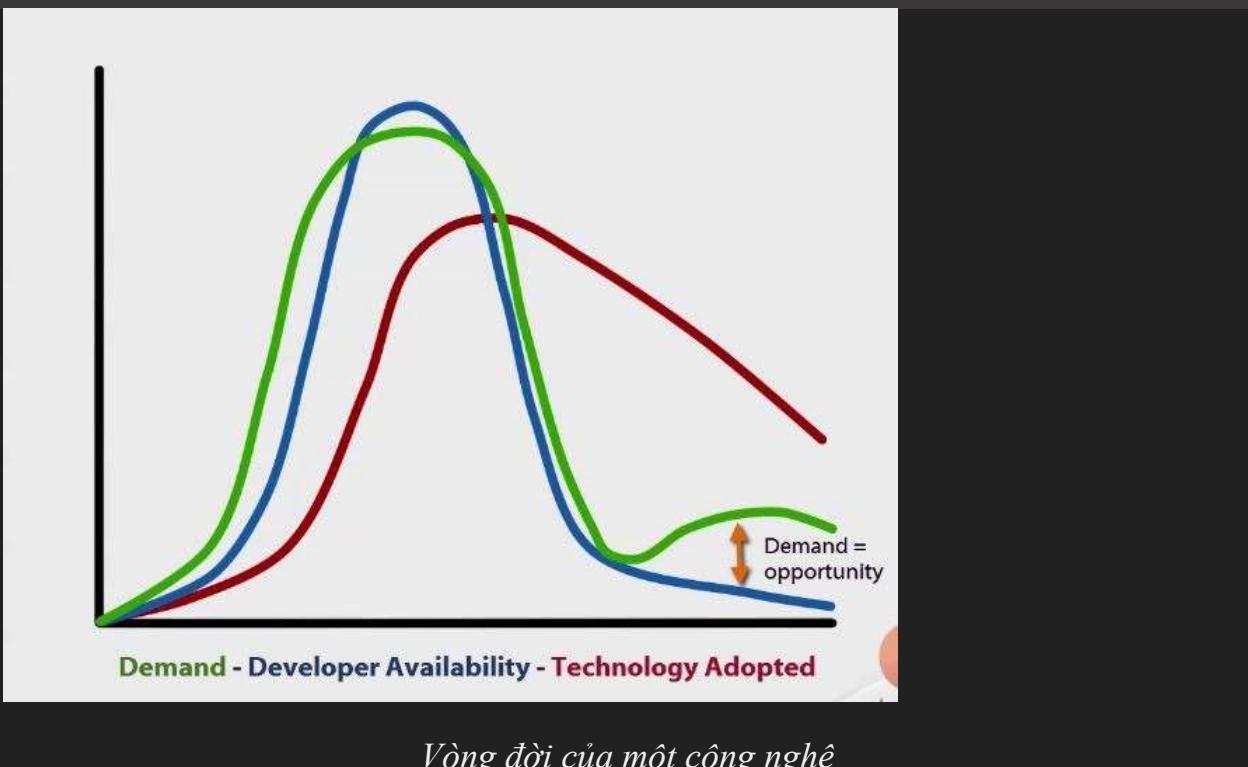
Nhìn chung, sự thay đổi này có mặt tích cực của nó: Các framework/library mới có nhiều tính năng hơn, giúp việc code nhanh và dễ dàng hơn. Tuy nhiên, điều đó cũng đi kèm không ích phiền toái: Mỗi version lại có chút ít cập nhật và thay đổi, làm việc nâng cấp/tích hợp rất mệt mỏi và mất thời gian. Một ví dụ điển hình là ngôn ngữ Python: Python có 2 phiên bản 2 và 3. Phiên bản 3 tích hợp quá nhiều thay đổi, nhiều thư viện của python 2 không chạy được trên bản 3, ... do đó cộng đồng Python vẫn tranh cãi về việc dùng phiên bản nào. Phiên bản lớn nhất chính là: **Tốn công sức, thời gian mà lập trình viên đã bỏ ra để học ngôn ngữ đó.**

Đôi khi một framework/library **chết**, không được hỗ trợ nữa (VB6, Silverlight). Thủ tướng tượng bây giờ MS SQL không được phát triển tiếp xem, một đồng dự án sử dụng MS SQL sẽ lao đao. Đó là lý do các ngôn ngữ/công nghệ như Java, C#, PHP, MySQL vẫn được ưa chuộng so với NodeJS, MongoDB, ... vì chúng có tuổi đời lâu hơn, đáng tin tưởng hơn.



Một số công nghệ đã “tắt thở” của Microsoft

Vòng đời của một công nghệ: Một công nghệ mới ra đời, được nhiều công ty sử dụng, nhu cầu tuyển dụng cao nên nhiều người theo học. Qua năm tháng, công nghệ chết dần, không ai tuyển nữa nên ít người học. Tuy nhiên, ứng dụng của các công ty được xây dựng bằng các công nghệ cũ, họ vẫn cần developer để bảo trì/nâng cấp ứng dụng. Đó là lý do các developer Cobol, VB6, Fortran vẫn rất được giá (Bọn Nhật có nhiều hệ thống lớn và siêu lớn xây dựng bằng các công nghệ này, toàn outsource cho dân VN bảo trì).



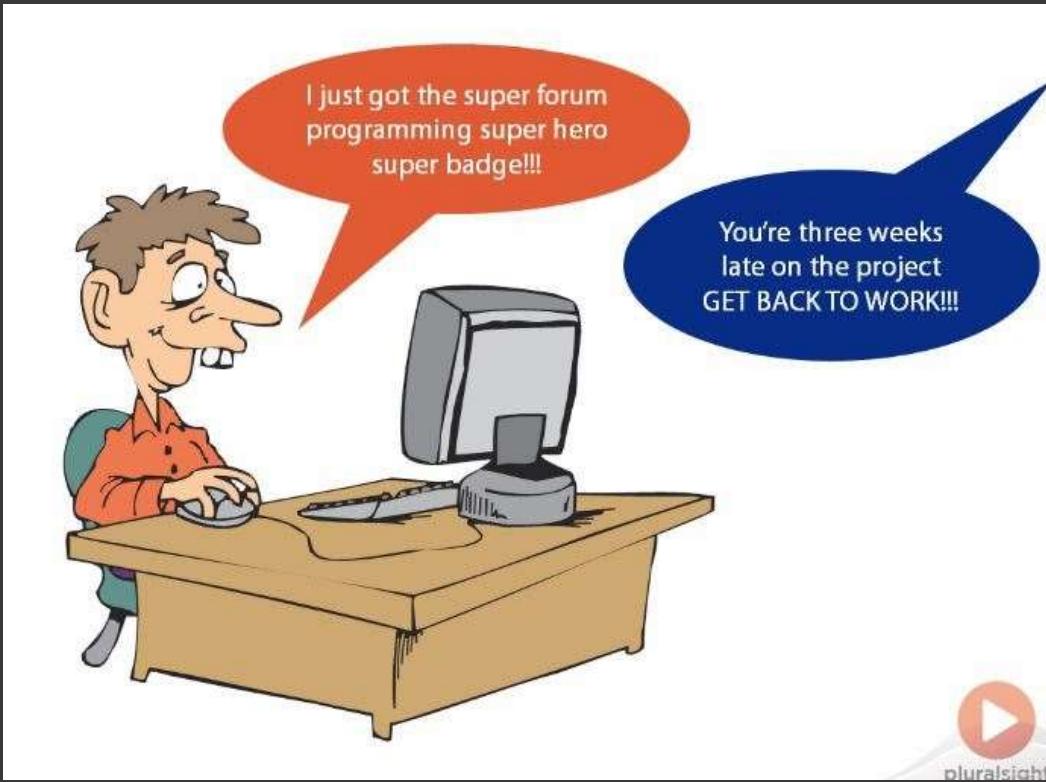
2. Hackathon và Stackoverflow không “tốt đẹp” như bạn nghĩ

Hackathon là một cuộc thi code nho nhỏ, thường tổ chức vào hai ngày cuối tuần. Các lập trình viên sẽ lập thành một team khoảng 3-5 người để xây dựng một sản phẩm. Giải thưởng (khá hấp dẫn) sẽ được trao cho các đội xuất sắc nhất.

Mặt tốt của hackathon rất nhiều và dễ thấy. Các sự kiện này thường rất vui và có ích cho lập trình viên: Họ được vui chơi, được code, được tặng quà, còn được xây dựng quan hệ, kết nối với cộng đồng. Tuy nhiên, **mặt xấu thì ít người để ý**. Hackathon được tổ chức với lý do là “phát triển cộng đồng, giới thiệu công nghệ”. Thật ra, đây là một cách rẽ mặt để **chôm thời gian và công sức của lập trình viên**. Luật bản quyền cũng ít khi được coi trọng (ứng dụng tham gia hackathon, mặc định bản quyền thuộc về hackathon) nên bạn có thể dễ dàng bị chôm/đạo ý tưởng mà không làm được gì.



Stackoverflow là một mạng hỏi-đáp dành cho lập trình viên. Tham gia [stackoverflow](#) sẽ giúp bạn nâng cao trình độ và kiến thức. Tuy nhiên, tham gia stackoverflow, bạn đã vô tình tham gia vào đội ngũ “chuyên gia” miễn phí, cho stackoverflow thu tiền quảng cáo. Nó còn áp dụng Gamification (Điểm, huy hiệu) để dụ dỗ bạn đóng góp thời gian, công sức. Một câu trả lời trên stackoverflow tiết kiệm cho bạn 1 tiếng fix lỗi, bạn bỏ ra 15 phút ngồi đóng góp lại cũng ko sao, nhưng đừng bỏ 3, 4 tiếng vào đó nhé.

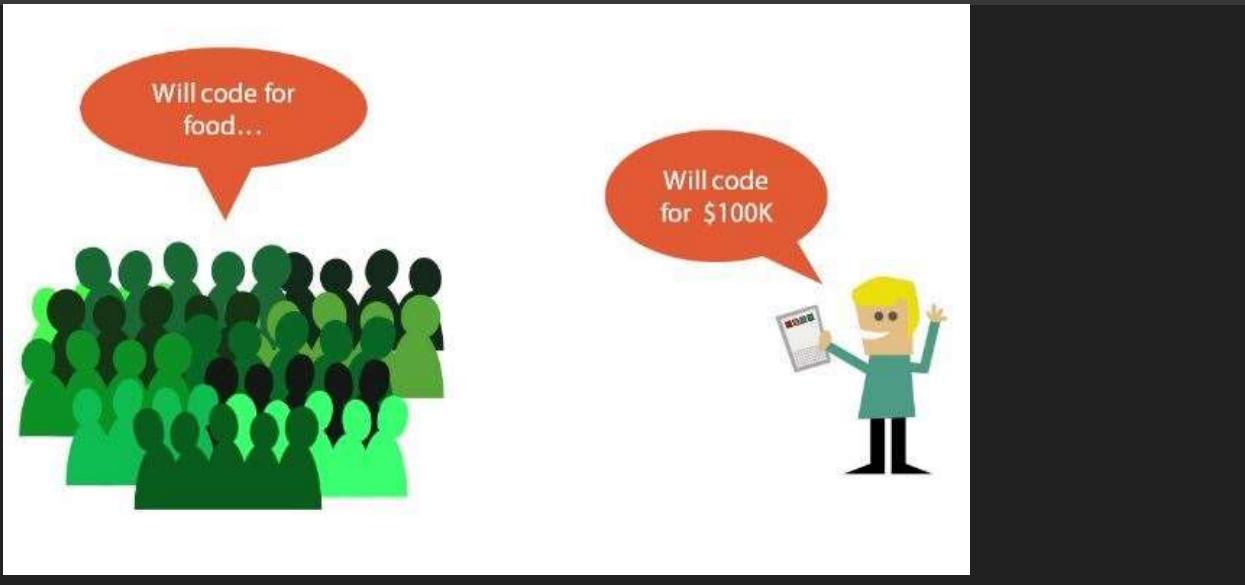


Dù vậy, mình vẫn khuyên các bạn tham dự các hackathon và stackoverflow, chúng rất hữu ích với mọi lập trình viên. Nhưng nhớ biết nhận thức và dè chừng những mặt xấu của nó nhé.

3. Vấn đề outsource và bóc lột

Ở Việt Nam có khá nhiều công ty outsource (FPT, Harvey Nash, ...). Các công ty này cung cấp phần lớn công ăn việc làm cho lập trình viên, đóng góp vào GDP nước nhà. Outsource cũng không phải hoàn toàn xấu, nhưng quá lệ thuộc vào outsource, chỉ cắm đầu vào công phần mềm, thì ngành IT nước nhà sẽ rất khó phát triển.

Vì sao các công ty nước ngoài lại outsource, chung quy cũng (lại) là **vì tiền**. Giá nhân sự của Việt Nam, Ấn Độ rẻ hơn lập trình viên nước ngoài nhiều. Các công ty outsource Việt Nam ăn phần lớn, trả một phần nhỏ cho lập trình viên. Đó là lý do các bác PM, quản lý, cấp cao lương thường cao chót vót, còn developer quèn thì lương nhỏ giọt. Mình khuyên các bạn nên học giỏi tiếng Anh rồi tự tìm cách bán thân – qua nước ngoài làm. Mình quen vài ông anh làm cho công ty nước ngoài, 1 ông qua Singapore làm, lương cao mà không bị bóc lột như outsource.



Outsource qua các nước nghèo với giá rẻ mạt

Nhắc tới chuyện bóc lột, hình thức bóc lột “dã man” nhất là **làm thêm giờ**, còn gọi là OT (Overtime). Nếu bạn xui xẻo rơi vào dự án “cháy”, OT là chuyện như cơm bữa. Bạn sẽ phải đi làm tới 8-9 giờ tối, cả thứ 7 chủ Nhật. Điều này ảnh hưởng không nhỏ đến sức khỏe, quan hệ xã hội và gia đình. Mình từng nghe “truyền thuyết” về 1 team cũ ở BU26 FPT, do OT ăn mì tôm riết mà cả nhóm bị mõi trong máu. Về chuyện tiền nong, có nhiều công ty trả tiền OT rất sòng phẳng, cũng có nhiều nơi quy ra ngày phép, hoặc xù luôn. **Làm với Nhật thì không cái khái niệm OT, đi làm 9-10h tối mới về là “văn hóa làm việc”** bên đó.

Chưa hết đâu, còn một số mặt tối đáng sợ hơn sẽ được đề cập trong [phần 2](#), các bạn đón đọc nhé.

Dependency Injection và Inversion of Control – Phần 3: DI Container. Áp dụng DI vào ASP.NET MVC

Posted on 12/11/2015 by Pham Huy Hoàng

Series bài viết Dependency Injection và Inversion of Control gồm 3 phần:

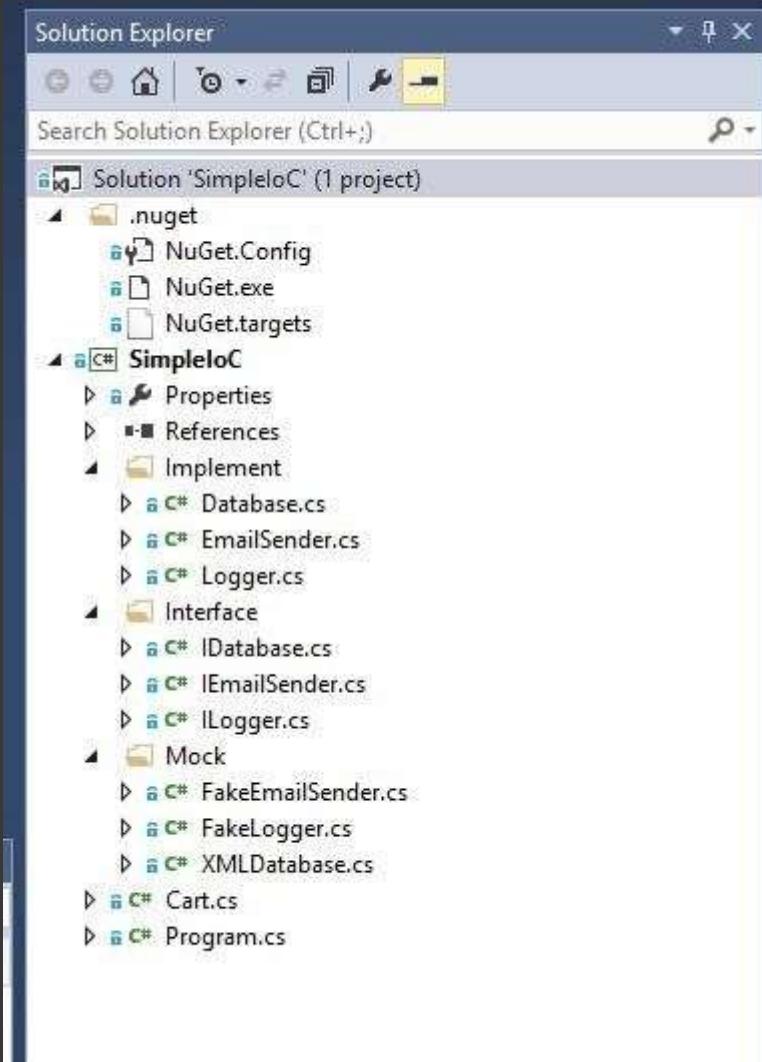
1. Định nghĩa
2. Áp dụng DI vào code
3. **Viết DI Container. Áp dụng DI vào ASP.NET MVC**

Sau 2 phần đầu, chắc các bạn đã có cái nhìn tổng quan về DI và cách áp dụng chúng vào code. Đa phần chúng ta không tự viết sử dụng các DI Container nổi tiếng như: Unity, NInject, StructureMap.

Để hiểu nguyên lý hoạt động của chúng, mình sẽ cùng các bạn cách viết một DI Container đơn giản (chúng cũng không quá “ghê gớm” hay phức tạp như bạn nghĩ đâu). Sau đó mình sẽ hướng dẫn cách sử dụng cái DI Container có sẵn, cũng như áp dụng IoC và project MVC.

1. Tự viết 1 DI Container đơn giản

Các bạn có thể dùng git để clone project về máy và bắt đầu làm theo
mình: <https://github.com/ToiDiCodeDaoSampleCode/SimpleIoC>. Các class và
interface vẫn như trong phần 2, có điều mình đã bổ sung thêm 1 số class mock –
module giả. Trong thực tế, ta sử dụng các class mock này để viết Unit Test.



DI Container thường có 1 function dùng để setup module và interface, một function khác để lấy module dựa theo interface. Ở đây mình gọi 2 function đó là **SetModule** và **GetModule**.

```
1     public class DIContainer
2     {
3         public static void SetModule<TInterface, TModule>()
4         {
5             SetModule(typeof(TInterface), typeof(TModule));
6         }
7
8         public static T GetModule<T>()
```

```

9         return (T)GetModule(typeof(T));
10    }
11 }
12

```

Code của hàm Main cũng rất đơn giản. Ta chỉ cài đặt các interface và module tương ứng thông qua function SetModule. Với class Cart, ta chỉ cần gọi hàm GetModule. DIContainer sẽ tự inject IDatabase, ILogger vào theo code ta đã viết.

```

1 //Với mỗi Interface, ta define một Module tương ứng
2 DIContainer.SetModule<IDatabase, Database>();
3 DIContainer.SetModule<ILogger, Logger>();
4 DIContainer.SetModule<IEmailSender, EmailSender>();
5
6 DIContainer.SetModule<Cart, Cart>();
7
8 //DI Container sẽ tự inject Database, Logger vào Cart
9 var myCart = DIContainer.GetModule<Cart>();

```

Class DI Container sẽ có các đặc tính sau:

- Lưu trữ các Interface, Module tương ứng vào một Dictionary có Key là Interface, Value là Module. Để lấy một Module từ Container, ta cần đưa vào Interface của Module đó.
- Khi cài đặt một module, container sẽ tìm Constructor đầu tiên của module đó.
- Nếu constructor không có tham số (Module không có dependency), container sẽ **gọi constructor này** để khởi tạo module.
- Nếu constructor này có tham số (Có dependency), container sẽ **khởi tạo các tham số này**, gán chúng vào constructor của module. Đây là quá trình injection.

Vì việc implement cũng không phức tạp lắm, bạn đọc code và comment sẽ hiểu thôi.

```

1 public class DIContainer
{

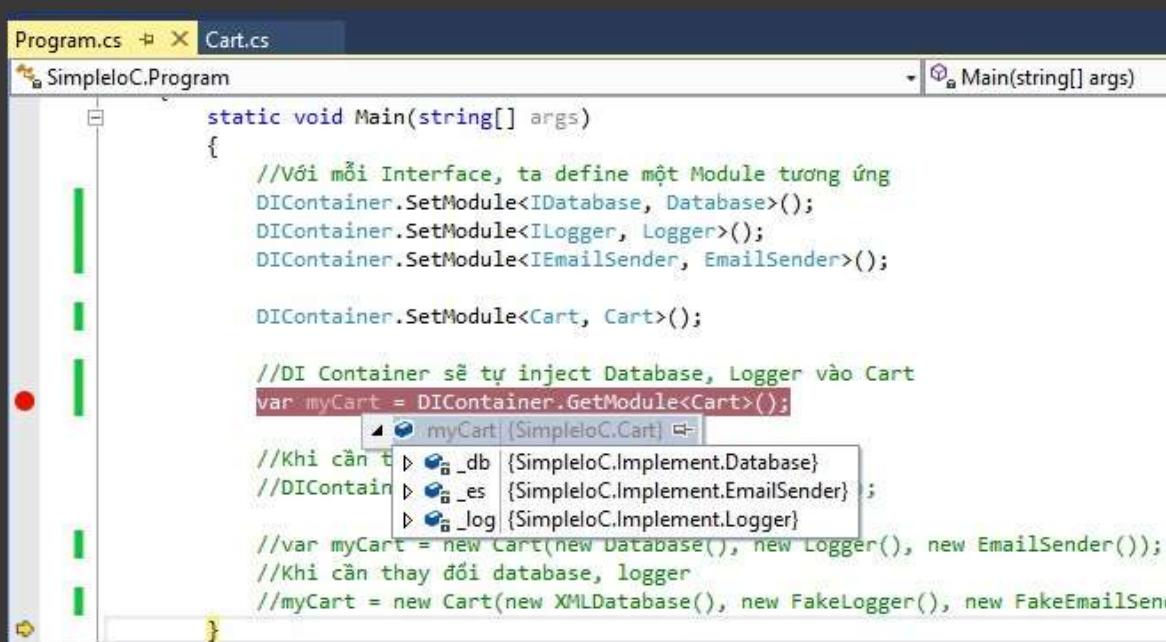
```



```
30         //Khởi tạo module
31         module = firstConstructor.Invoke(null); // new Database(), new Logger()
32     }
33     else
34     {
35         //Lấy các tham số của constructor
36         var constructorParameters = firstConstructor.GetParameters(); //IDatabase
37
38         var moduleDependecies = new List<object>();
39         foreach (var parameter in constructorParameters)
40         {
41             var dependency = GetModule(parameter.ParameterType); //Lấy module tu
42             moduleDependecies.Add(dependency);
43
44             //Inject các dependency vào constructor của module
45             module = firstConstructor.Invoke(moduleDependecies.ToArray());
46
47             //Lưu trữ interface và module tương ứng
48             ResgisteredModules.Add(interfaceType, module);
49
50         }
51
52         private static object GetModule(Type interfaceType)
53         {
54             if (ResgisteredModules.ContainsKey(interfaceType))
55             {
56                 return ResgisteredModules[interfaceType];
57             }
58             throw new Exception("Module not register");
59         }
60     }
61 }
```

58
59
60
61
62
63
64

Kết quả:



```
Program.cs ➔ X Cart.cs
SimpleIoC.Program
Main(string[] args)
static void Main(string[] args)
{
    //Với mỗi Interface, ta define một Module tương ứng
    DIContainer.SetModule<IDatabase, Database>();
    DIContainer.SetModule<ILogger, Logger>();
    DIContainer.SetModule<IEmailSender, EmailSender>();

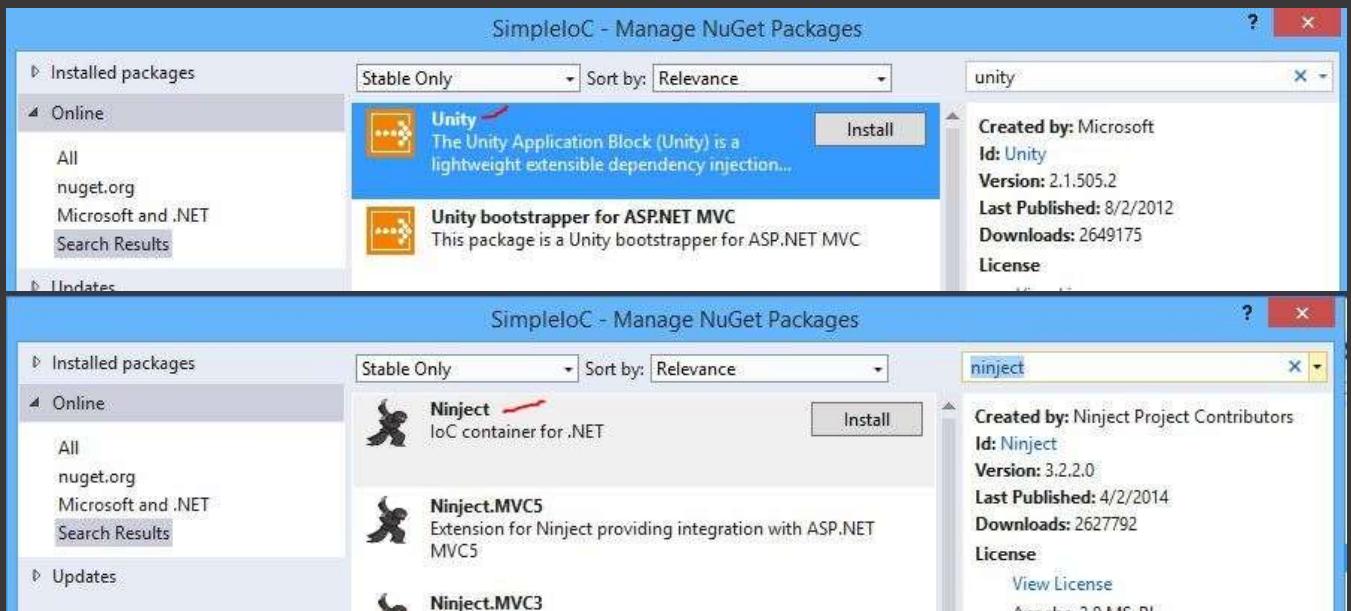
    DIContainer.SetModule<Cart, Cart>();

    //DI Container sẽ tự inject Database, Logger vào Cart
    var myCart = DIContainer.GetModule<Cart>();
    myCart (SimpleIoC.Cart) ↴
        ▾ _db {SimpleIoC.Implement.Database}
        ▾ _es {SimpleIoC.Implement.EmailSender}
        ▾ _log {SimpleIoC.Implement.Logger}
    //var myCart = new Cart(new Database(), new Logger(), new EmailSender());
    //Khi cần thay đổi database, logger
    //myCart = new Cart(new XMLDatabase(), new FakeLogger(), new FakeEmailSender());
}
```

2. Sử dụng DI Container từ các framework có sẵn

Tất nhiên, nếu người khác đã viết sẵn, kiểm thử và fix lỗi, chúng ta có thể tái sử dụng mà không cần phải viết lại từ đầu cho mệt. Mình sẽ hướng dẫn các bạn sử dụng DI Container của *Ninject* và *Unity*.

Dùng Nuget (hoặc Package Manager Console) để cài đặt *Ninject* và *Unity*. Nhấp chuột phải vào project SimpleIoC, chọn Manage Nuget packages.



Vì bạn đã chia code ra thành các module và interface rồi, ta chỉ cần thay code của DIContainer thành code Unity và Ninject là được.

Ninject

```

1  var kernel = new StandardKernel();
2  kernel.Bind<IDatabase>().To<Database>();
3  kernel.Bind<ILogger>().To<Logger>();
4  kernel.Bind<IEmailSender>().To<EmailSender>();
5  kernel.Bind<Cart>().To<Cart>();
6
7  //DI Container sẽ tự inject Database, Logger vào Cart
8  var myCart = kernel.Get<Cart>();

```

Unity

```

1  var container = new UnityContainer();
2  container.RegisterType<IDatabase, Database>();
3  container.RegisterType<ILogger, Logger>();
4  container.RegisterType<IEmailSender, EmailSender>();
5  container.RegisterType<Cart, Cart>();

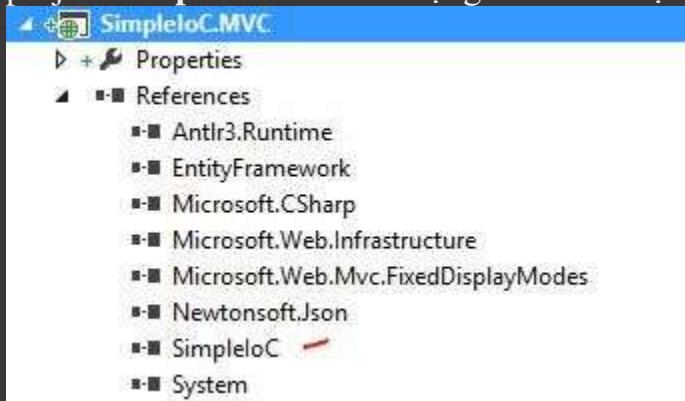
```

```
7 //DI Container sẽ tự inject Database, Logger vào Cart  
8 var myCart = container.Resolve<Cart>();
```

3. Áp dụng IoC vào project MVC

Đa phần các framework như Spring.NET, ASP.NET MVC, Spring, Struts đều có sẵn DI Container, hoặc cho phép tích hợp DI Container bên ngoài. Vì đây là blog C# nên mình sẽ hướng dẫn cách **tích hợp IoC vào project MVC**. Mọi công đoạn chỉ mất từ 2-5 phút. (**WebForm không áp dụng IoC** được vì Page Cycle khá phức tạp, không cho phép ta can thiệp vào quá trình khởi tạo page).

Bước 1. Tạo 1 project MVC trong cùng Solution. Add References tới project **SimpleIoC** để tái sử dụng class cho tiện.



Bước 2. Thêm parameter vào constructor của controller, module **Cart** sẽ được inject vào. Nếu chạy thử bạn sẽ bị lỗi “No parameterless constructor ...”

```

public class HomeController : Controller
{
    //
    // GET: /Home/
    private Cart _cart;

    public HomeController(Cart cart)
    {
        _cart = cart;
    }

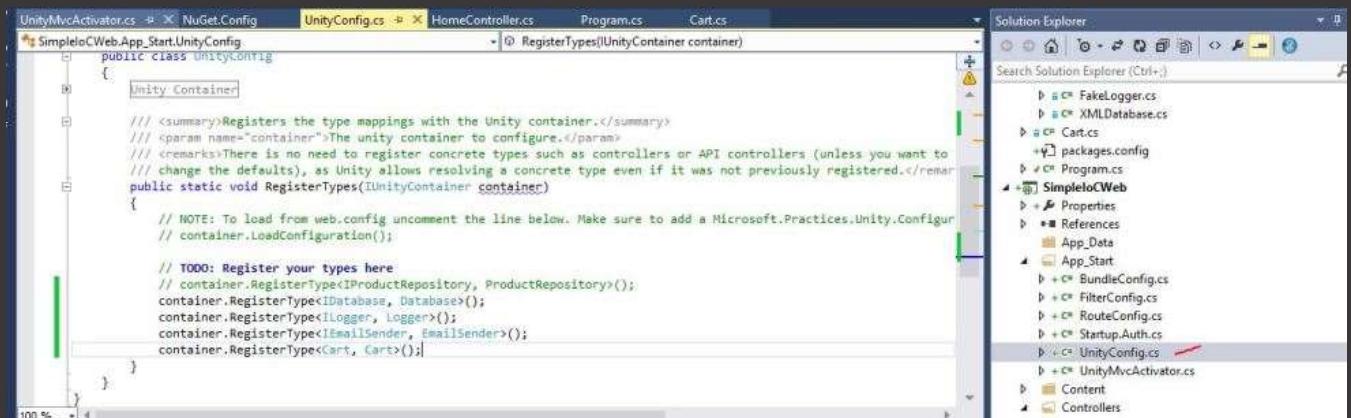
    public ActionResult Index()
    {
        return View();
    }
}

```



Bước 3. Sử dụng Nuget, tìm và cài **Unity Bootstraper for ASP.NET MVC** (Các DI Container khác như *Ninject*, *StructureMap* đều có Bootstraper cho MVC cả, đừng lo). Ta sẽ thấy có file mới tên là **UnityConfig** được tạo ra.

Ta vào file này và thực hiện việc setup các interface và module. Tùy theo phiên bản MVC mà nội dung các file tạo ra có khác nhau chút đỉnh. Tuy nhiên cú pháp cài đặt vẫn như cũ.



Bước 4. Chạy thử và ... xong. Chúc mừng bạn đã hoàn thành bài viết 3 phần về IoC đầy khó khăn và gian khổ.



```
namespace SimpleIoCWeb.Controllers
{
    public class HomeController : Controller
    {
        private Cart _cart;
        public HomeController(Cart cart)
        {
            _cart = cart;
        }
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult About()
        {
            ViewBag.Message = "Your application description page.";
            return View();
        }
    }
}
```

Nhờ IoC và mock, ta có thể Unit Test từng module riêng lẻ. Các bạn có thể xem thêm ở đây: <https://duyphuong13.wordpress.com/2013/12/15/mot-so-ky-thuat-trong-unit-test/>, vì có bạn đã viết khá chi tiết rồi nên mình không viết lại nữa. Nếu có góp ý hay gạch đá gì các bạn cứ ném thoải mái trong phần comment nhé.

Dependency Injection và Inversion of Control – Phần 2: Áp dụng DI vào code

Posted on 10/11/2015 by Phạm Huy Hoàng

Series bài viết Dependency Injection và Inversion of Control gồm 3 phần:

1. [Định nghĩa](#)
2. **Áp dụng DI vào code**
3. [Viết DI Container. Áp dụng DI vào ASP.NET MVC](#)

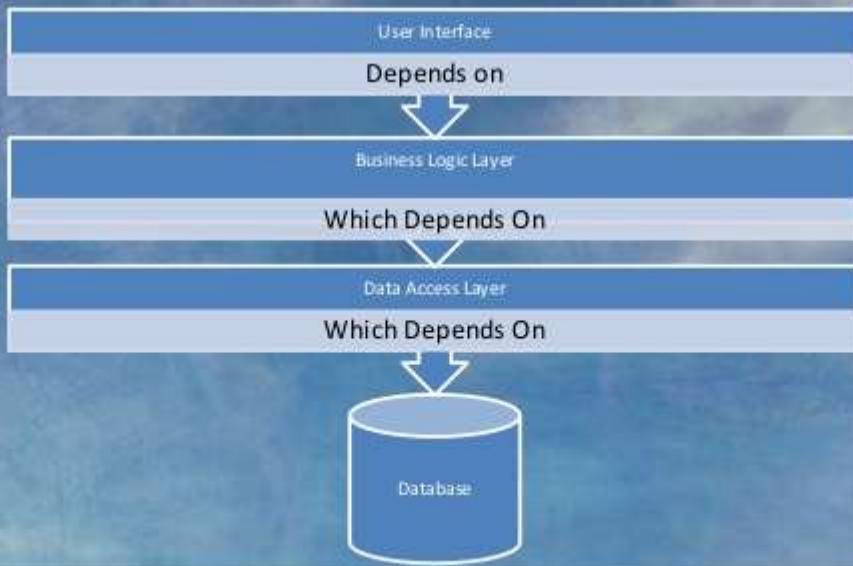
Bạn đã đọc [phần 1](#) nhưng vẫn *chưa hiểu rõ lắm* về DI, IoC, chưa biết cách áp dụng chúng vào code? Đừng lo, ở phần 2 này sẽ cung cấp những đoạn code mẫu, **giải thích rõ hơn** về những điều mình đã nói ở phần 1. Sau khi đọc xong phần này, các bạn quay lại phần 1 thì sẽ thấy “thông” ra được nhiều thứ nhé.

Dependency là gì?

Dependency là những module cấp thấp, hoặc cái service gọi từ bên ngoài. Với cách code thông thường, các module cấp cao sẽ gọi các module cấp thấp. Module cấp cao **sẽ phụ thuộc và module cấp thấp**, điều đó tạo ra các dependency.

Dependencies

Lower application layers
External services
Other components



Để dễ hiểu, hãy xem hàm Checkout của class Cart dưới đây. Hàm này sẽ lưu order xuống database và gửi email cho user. Class Cart sẽ khởi tạo và gọi module Database, module EmailSender, module Logger, các module này chính là các dependency.

```
1     public class Cart
2     {
3         public void Checkout(int orderId, int userId)
4         {
5             Database db = new Database();
6             db.Save(orderId);
7
8             Logger log = new Logger();
9             log.LogInfo("Order has been checkout");
```

9

```
10     EmailSender es = new EmailSender();  
11     es.SendEmail(userId);  
12 }  
13 }  
14 }
```

Cách làm này có gì sai không? Có vẻ là không, viết code cũng nhanh nữa. Nhưng cách viết này “có thể” sẽ dẫn tới một số vấn đề trong tương lai:

- Rất khó test hàm Checkout này, vì nó dính dáng tới cả hai module Database và EmailSender.
- Trong trường hợp ta muốn thay đổi module Database, EmailSender,... ta phải sửa toàn bộ các chỗ khởi tạo và gọi các module này. Việc làm này rất mất thời gian, dễ gây lỗi.
- Về lâu dài, code sẽ trở nên “kết dính”, các module có tính kết dính cao, một module thay đổi sẽ kéo theo hàng loạt thay đổi. Đây là nỗi ác mộng khi phải maintainance code.

Inversion of Control và Dependency Injection đã ra đời để giải quyết những vấn đề này.

Làm sao để hạn chế coupling giữa các class. Đã có Inversion of Control
Để các module không “kết dính” với nhau, chúng không được **kết nối trực tiếp**, mà
phải **thông qua interface**. Đó cũng là nguyên lý cuối cùng trong SOLID.

1. Các module cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả 2 nên phụ
thuộc vào abstraction.

2. Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại. (Các class
giao tiếp với nhau thông qua interface, không phải thông qua implementation.)

Ta lần lượt tạo các interface IDatabase, IEmailSender, ILogger, các class kia ban đầu
sẽ lần lượt kế thừa những interface này. Để dễ hiểu, giờ mình sẽ tạm gọi *IDatabase*,
IEmailSender, *ILogger* là **Interface**, các class như *Database*, *EmailSender*, *Logger*
là **Module**.


```
29     }
30
31     public class EmailSender : IEmailSender
32     {
33         public void SendEmail(int userId)
34         {
35             //...
36         }
37
38
39
40
```

Hàm checkout mới sẽ trông như sau:

```
1     public void Checkout(int orderId, int userId)
2     {
3         // Nếu muốn thay đổi database, ta chỉ cần thay dòng code dưới
4         // Các Module XMLDatabase, SQLDatabase phải implement IDatabase
5         //IDatabase db = new XMLDatabase();
6         //IDatabase db = new SQLDatabase();
7         IDatabase db = new Database();
8         db.Save(orderId);
9
10        ILogger log = new Logger();
11        log.LogInfo("Order has been checkout");
12
13        IEmailSender es = new EmailSender();
14        es.SendEmail(userId);
15    }
```

Với interface, ta có thể dễ dàng thay đổi, swap các module cấp thấp mà **không ảnh hưởng tới module Cart**. Đây là *bước đầu của IoC*.

Để dễ quản lý, ta có thể bỏ tất cả những hàm khởi tạo module vào constructor của class Cart.

```
1
2     public class Cart
3     {
4         private readonly IDatabase _db;
5         private readonly ILogger _log;
6         private readonly IEmailSender _es;
7
8         public Cart()
9         {
10            _db = new Database();
11            _log = new Logger();
12            _es = new EmailSender();
13
14            public void Checkout(int orderId, int userId)
15            {
16                _db.Save(orderId);
17                _log.LogInfo("Order has been checkout");
18                _es.SendEmail(userId);
19            }
20        }
```

Cách này thoạt nhìn khá khá ổn. Tuy nhiên, nếu có nhiều module khác cần dùng tới Logger, Database, ta lại phải khởi tạo các Module con ở constructor của module đó. Có vẻ không ổn phải không nào?

Dependencies Inversion (Principle)

Inversion of Control (Pattern)

Service Locator

Event

Delegate

**Dependency
Injection**

Ban đầu, người ta dùng **ServiceLocator** để giải quyết vấn đề này. Với mỗi Interface, ta set một Module tương ứng. Khi cần dùng, ta sẽ lấy Module đó từ ServiceLocator. Đây cũng là một cách để hiện thực IoC.

```
1     public static class ServiceLocator
2     {
3         public static T GetModule()
4         {
5             //....
6         }
7
8         //Ta chỉ việc gọi hàm GetModule
9         public class Cart
10        {
```

```
11     public Cart()
12     {
13         _db = ServiceLocator.GetModule();
14         _log = ServiceLocator.GetModule();
15         _es = ServiceLocator.GetModule();
16     }
17
18
```

Cách này vẫn còn khuyết điểm: *toàn bộ các class đều phụ thuộc vào ServiceLocator.* **Dependency Injection** giải quyết được vấn đề này. Các **Module cấp thấp sẽ được inject (truyền vào) vào Module cấp cao** thông qua Constructor hoặc thông qua Properties. Nói một cách đơn giản dễ hiểu về DI:

Ta không gọi toán tử new để khởi tạo instance, mà instance đó sẽ được truyền từ ngoài vào (Truyền manual, hoặc nhờ DI Container).



Sau khi áp dụng Dependency Injection, ta sẽ sử dụng class Cart như sau:

```
1  public Cart(IDatabase db, ILogger log, IEmailSender es)
2  {
3      _db = db;
4      _log = log;
5      _es = es;
6
7      //Dependency Injection một cách đơn giản nhất
8      Cart myCart = new Cart(new Database(),
9                          new Logger(), new EmailSender());
10     //Khi cần thay đổi database, logger
11     myCart = new Cart(new XMLDatabase(),
```

```
12     new FakeLogger(), new FakeEmailSender());  
13
```

Chắc bạn nghĩ: Sau khi dùng Dependency Injection thì cũng phải khởi tạo Module à, thế thì còn dở hơn ServiceLocator rồi. Thông thường, ta sử dụng DI Container. Chỉ việc define **một lần**, DI Container sẽ **tự thực hiện việc inject** các module cấp thấp vào module cấp cao.

```
1 //Với mỗi Interface, ta define một Module tương ứng  
2 DIContainer.SetModule<IDatabase, Database>();  
3 DIContainer.SetModule<ILogger, Logger>();  
4 DIContainer.SetModule<IEmailSender, EmailSender>();  
5  
6 DIContainer.SetModule<Cart, Cart>();  
7  
8 //DI Container sẽ tự inject Database, Logger vào Cart  
9 var myCart = DIContainer.GetModule();  
10  
11 //Khi cần thay đổi, ta chỉ cần sửa code define  
12 DIContainer.SetModule<IDatabase, XMLDatabase>();
```

Sau khi áp dụng Dependency Injection, code bạn sẽ dài hơn, có vẻ “phức tạp” hơn và sẽ *khó debug hơn*. Đổi lại, code sẽ **uyển chuyển, dễ thay đổi** cũng như **dễ test hơn**. Như mình đã nói ở bài trước, **không phải lúc nào DI cũng là lựa chọn phù hợp**, ta cần cân nhắc các ưu khuyết điểm. DI được áp dụng trong nhiều framework back-end (ASP.MVC, Struts2) lẫn front-end (AngularJS, KnockoutJS). Đa phần các dự án lớn trong các công ty IT đều áp dụng DI, do đó những kiến thức về DI sẽ rất hữu ích khi phỏng vấn cũng như làm việc.



Dependency Injection

Two notations to inject

- ❖ From parameter names in functions:

```
app.controller('SuperheroicController', function($scope){  
  $scope.hello = 'world';  
});
```

- ❖ Inline array notation:

```
app.controller('SuperheroicController', ['$scope', function(whatever)  
{  
  whatever.hello = 'world';  
}]);
```



T3DD14
The Test Driven Development Conference

sunzinet⁺

20th June 2014, Armin Buddeberg Viewed 18,691 times
Getting started with super heroic JavaScript library Angular.js

Vậy cái DI Container phía trên **ở đâu ra**? Ta có thể tự viết, hoặc sử dụng một số DI Container phổ biến trong C# như: *Unity*, *StructureMap*, *NInject*. Ở [phần 3](#), mình sẽ hướng dẫn cách viết 1 DI Container đơn giản và dùng các DI Container sẵn có nhé.

Dependency Injection và Inversion of Control – Phần 1: Định nghĩa

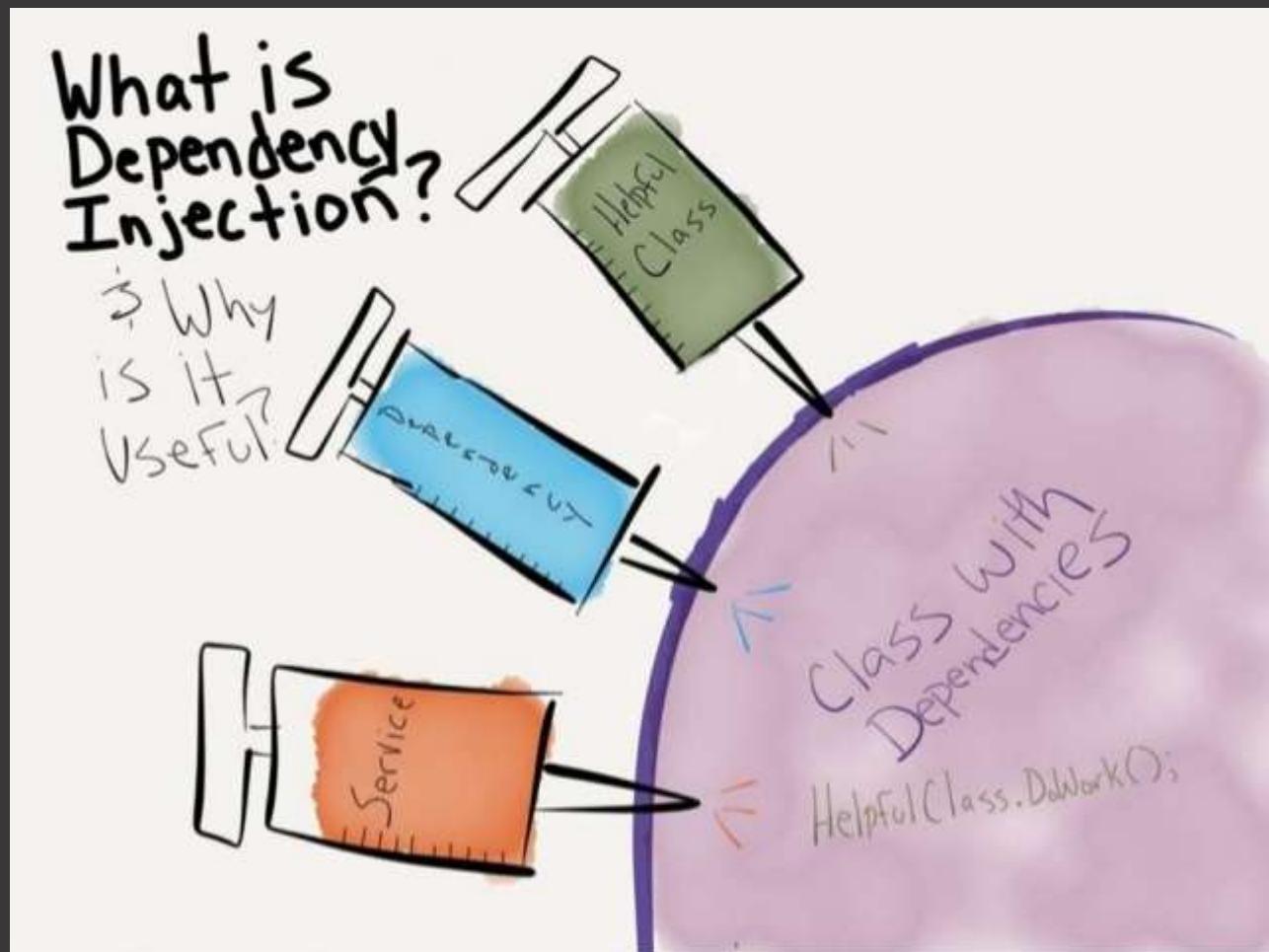
Posted on 03/11/2015 by Phạm Huy Hoàng

Series bài viết Dependency Injection và Inversion of Control gồm 3 phần:

1. [**Định nghĩa**](#)
2. [Áp dụng DI vào code](#)
3. [Viết DI Container. Áp dụng DI vào ASP.NET MVC](#)

Hôm trước, có vài bạn nhò mình giải thích các khái niệm **Dependency Injection**, **Inversion of Control**. Vốn lười, mình định google một bài viết bằng tiếng Việt để quăng cho các bạn ấy. Ngặc một nỗi là mình **chả tìm được bài nào** cụ thể rõ ràng về Dependency Injection, chỉ có hướng dẫn sử dụng Unity, StructureMap. Một số bài viết hay thì lại [toàn bằng tiếng Anh](#).

Mình cũng thấy vài bạn đã đi làm 1, 2 năm mà vẫn còn “ngáo ngo” về DI, IoC, chỉ biết sử dụng nhưng không hiểu rõ bản chất của nó. Do đó, mình viết bài này nhằm giải thích một cách đơn giản dễ hiểu về Dependency Injection. Các bạn [junior](#) nên đọc thử, vì DI được áp dụng rất nhiều trong các ứng dụng doanh nghiệp, rất hay gặp khi đi làm/đi phỏng vấn. Pattern này được dùng trong cả C#, Java và các ngôn ngữ khác nên các bạn cứ đọc thoải mái nhé.



Trong bài, mình hay dùng từ module. Trong thực tế, module này có thể là 1 project, 1 file dll, hoặc một service. Để dễ hiểu, **chỉ trong bài viết này, các bạn hãy xem mỗi module là một class** nhé.

Nhắc lại kiến thức

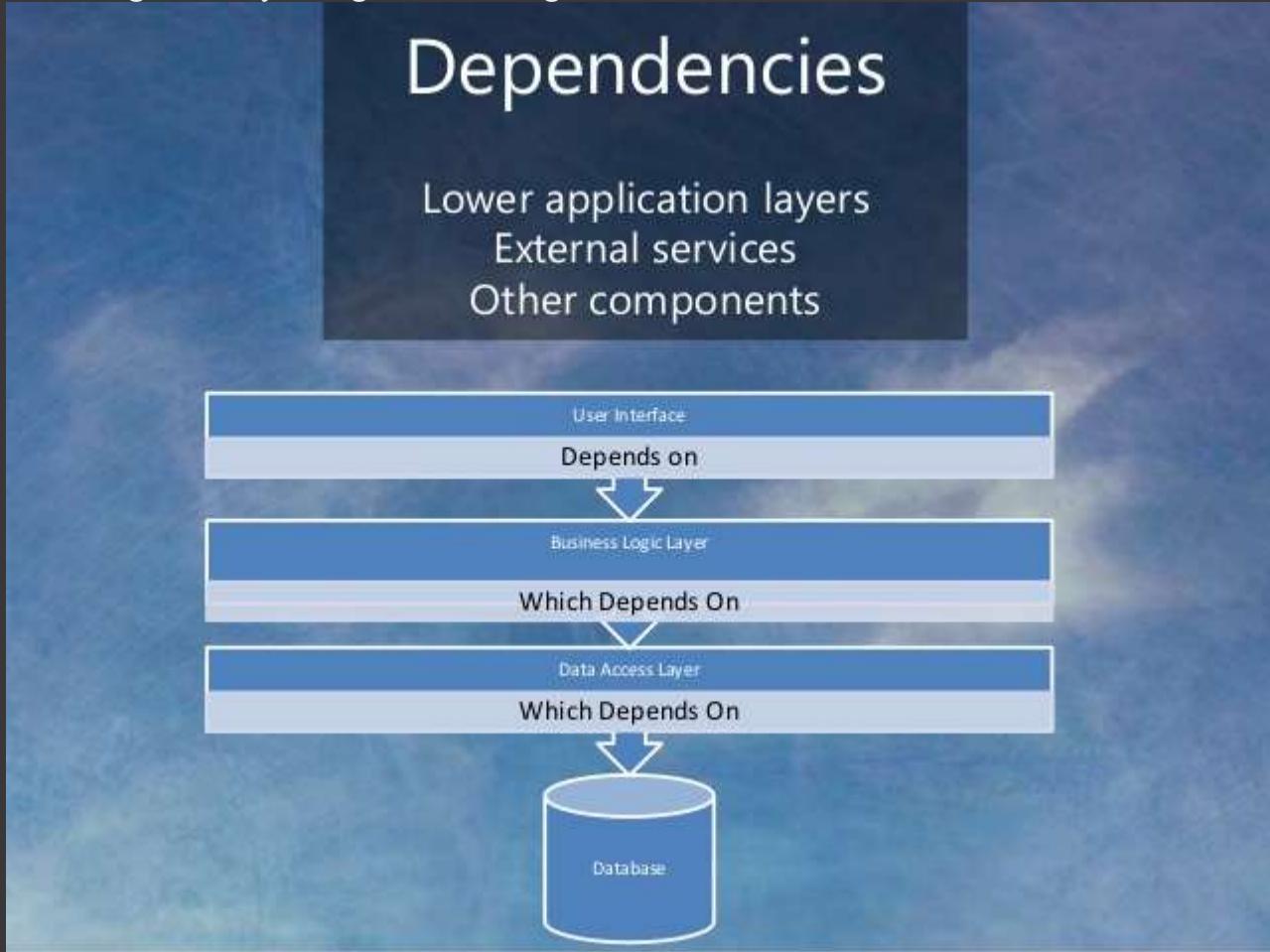
Trước khi bắt đầu với Dependency Injection, các bạn hãy đọc lại bài viết về [SOLID principles](#), những nguyên lý thiết kế và viết code. Nguyên lý cuối cùng trong SOLID chính là **Dependency Inversion**:

1. Các module cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả 2 nên phụ thuộc vào abstraction.

2. Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại. (Các class giao tiếp với nhau thông qua interface, không phải thông qua implementation.)

Với cách code thông thường, các module cấp cao sẽ gọi các module cấp thấp. Module cấp cao sẽ **phụ thuộc và module cấp thấp**, điều đó tạo ra các dependency. Khi

module cấp thấp thay đổi, module cấp cao **phải thay đổi theo**. Một thay đổi sẽ kéo theo hàng loạt thay đổi, giảm khả năng bảo trì của code.



Nếu tuân theo Dependency Inversion principle, các module cùng phụ thuộc vào 1 interface không đổi. Ta có thể dễ dàng thay thế, sửa đổi module cấp thấp mà không ảnh hưởng gì tới module cấp cao.

Định nghĩa và khái niệm DI

Hiện nay, các lập trình viên hay lầm lộn giữa các khái niệm Dependency Inversion, Inversion of Control (IoC), Dependency Injection (DI). Ba khái niệm này tương tự nhau nhưng không hoàn toàn giống nhau.

Dependencies Inversion (Principle)

Inversion of Control (Pattern)

Service Locator

Event

Delegate

**Dependency
Injection**

Sự khác biệt giữa 3 khái niệm trên:

- **Dependency Inversion:** Đây là một nguyên lý để thiết kế và viết code.
- **Inversion of Control:** Đây là một design pattern được tạo ra để code có thể tuân thủ nguyên lý Dependency Inversion. Có nhiều cách hiện thực pattern này: ServiceLocator, Event, Delegate, ... Dependency Injection là một trong các cách đó.
- **Dependency Injection:** Đây là một cách để hiện thực Inversion of Control Pattern (Có thể coi nó là một design pattern riêng cũng được). Các **module phụ thuộc (dependency) sẽ được inject vào module cấp cao**.

Khi nói tới DI, tức là nói tới Dependency Injection. Hiện nay, một số DI container như Unity, StructureMap v...v, hỗ trợ chúng ta trong việc cài đặt và áp dụng Dependency Injection vào code (Sẽ nói ở bài sau), tuy nhiên vẫn có thể gọi chúng là IoC Container, ý nghĩa tương tự nhau.

Có thể hiểu Dependency Injection một cách đơn giản như sau:

1. **Các module không giao tiếp trực tiếp với nhau, mà thông qua interface.**
Module cấp thấp sẽ implement interface, module cấp cao sẽ gọi module cấp thấp. Ví dụ: Để giao tiếp với database, ta có interface `IDatabase`, các module cấp thấp là `XMLDatabase`, `SQLDatabase`. Module cấp cao là `CustomerBusiness` sẽ sử dụng interface `IDatabase`.

2. Việc khởi tạo các module cấp thấp sẽ do DI Container thực hiện. Ví dụ:
Trong module CustomerBusiness, ta sẽ không khởi tạo `IDatabase db = new XMLDatabase()`, việc này sẽ do DI Container thực hiện. Module CustomerBusiness sẽ không biết gì về module XMLDatabase hay SQLDatabase.
3. Việc Module nào gắn với interface nào sẽ được config trong code hoặc trong file XML.
4. DI được dùng để làm giảm sự phụ thuộc giữa các module, dễ dàng hơn trong việc thay đổi module, bảo trì code và testing.

Các dạng DI

Có 3 dạng Dependency Injection:

1. **Constructor Injection:** Các dependency sẽ được container **truyền vào (inject vào)** 1 class thông qua constructor của class đó. Đây là cách thông dụng nhất.
2. **Setter Injection:** Các dependency sẽ được truyền vào 1 class thông qua các hàm Setter.
3. **Interface Injection:** Class cần inject sẽ implement 1 interface. Interface này chứa 1 hàm tên *Inject*. Container sẽ injection dependency vào 1 class thông qua việc gọi hàm *Inject* của interface đó. Đây là cách rườm rà và ít được sử dụng nhất.

```

3 references
public class InjectedCart : Cart
{
    private IDatabase _database;
    private IEmailSender _emailSender;      (new Database(), new EmailSender(), new XMLLogger());
    private ILogger _logger;

    //Constructor Injection
    public InjectedCart(IDatabase database, IEmailSender emailSender, ILogger logger)
    {
        _database = database;
        _emailSender = emailSender;
        _logger = logger;
    }

    //Properties Injection
    public ILogger Logger
    {
        set
        {
            _logger = value;
        }
    }
}

```

Ưu điểm và khuyết điểm của DI

Đĩ nhiên, DI không phải vạn năng, nó cũng có những ưu điểm và khuyết điểm, do đó không phải project nào cũng nên áp dụng DI. Với những dự án lớn, code nhiều, DI là thứ **rất cần thiền** để đảm bảo code dễ bảo trì, dễ thay đổi. Vì vậy, bản thân các framework nổi tiếng như Spring, Struts2, ASP.NET MVC, ... đều hỗ trợ hoặc tích hợp sẵn DI. ASP.NET MVC từ bản 5 trở xuống cho phép ta sử dụng DI container từ thư viện, từ bản 6 thì tích hợp sẵn DI luôn, không cần phải thêm thư viện gì.

Ưu điểm

- Giảm sự kết dính giữa các module
- Code dễ bảo trì, dễ thay thế module
 - Rất dễ test và viết Unit Test
 - Dễ dàng thấy quan hệ giữa các module (Vì các dependency đều được inject vào constructor)

Khuyết điểm

- Khái niệm DI khá “khó tiêu”, các developer mới sẽ gặp khó khăn khi học
- Sử dụng interface nên đôi khi sẽ khó debug, do không biết chính xác module nào được gọi
- Các object được khởi tạo toàn bộ ngay từ đầu, có thể làm giảm performance
- Làm tăng độ phức tạp của code

Bài viết khá nặng về lý thuyết nên *nếu bạn vẫn chưa mường tượng được sẽ áp dụng DI như thế nào vào code cũng đừng lo*. Ở phần 2 mình sẽ bổ sung code minh họa, các bạn đọc xong quay lại đọc bài này sẽ dễ “thông” hơn. Nhớ đón xem nhé.

Bonus: Một slide về Dependency Injection mình từng làm cho buổi thuyết trình trên công ty

<http://www.slideshare.net/conanak99/ioc-and-mapper-in-c>

Series C# hay ho: Tại sao WinForm vẫn “chưa chết” – Có nên học WinForm hay không ?

Posted on 03/12/2015 by Phạm Huy Hoàng

WinFom là một công nghệ của Microsoft, cho phép lập trình các ứng dụng Windows. Nhờ tính tiện ích, dễ code, giao diện design kéo thả đơn giản, ... Win Form đã được sử dụng để phát triển rất nhiều ứng dụng. Vì tuổi đời đã khá cao (ra đời vào năm 2003) nên WinForm chính thức bị Microsoft khai tử vào năm 2014. Tuy nhiên, ở Việt Nam, Win Form vẫn còn sống khỏe, **sống tốt**. Tại sao vậy? Thử đọc bài viết để biết nhé.

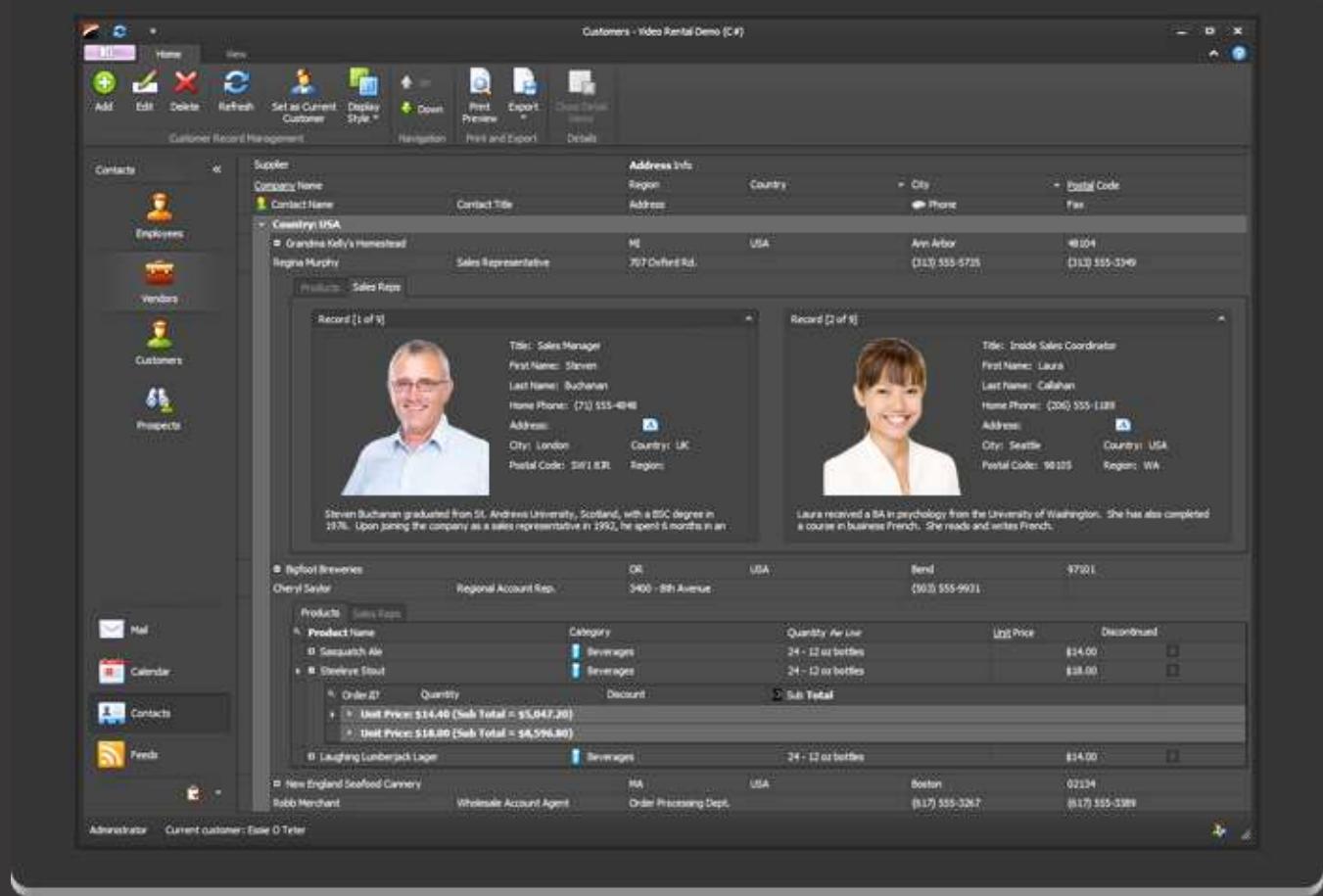


WinForm, người bạn của tuổi thơ – đơn giản mà hiệu quả

Đa phần lập trình viên C#. NET nào cũng từng học/sử dụng Win Form. Thuở còn ngồi trên ghế nhà trường, mình được tiếp xúc với ngôn ngữ C# lần đầu vào năm thứ hai, vài ngày sau đó là WinForm. Thực sự, mình rất ám tượng khi lần đầu tiếp xúc với WinForm: Giao diện kéo thả dễ sử dụng; Gắn các event cho các button chỉ cần double click, lại hỗ trợ quá trời event như click, hover,...; Việc viết code cũng vô cùng trực quan: từ việc lấy text từ TextBox cho tới show dữ liệu bằng MessageBox, hoặc dùng Grid để kết nối SQL. Đó là một trong các lý do mà nó được các trường/sinh viên ưa thích: **WinForm rất dễ học và dễ dạy**.

Vì dễ code, chỉ cần kéo thả, lại có nhiều component có sẵn, WinForm rất phù hợp để làm các phần mềm quản lý, tính tiền, thống kê... . Đây cũng là loại ứng dụng mà các

công ty/doanh nghiệp vừa và nhỏ cần. Ngoài ra, chỉ cần **sử dụng component như TelerikUI hoặc DevExpress** (Ở Việt Nam, hầu như chúng ta đều dùng crack nên các component này hoàn toàn miễn phí), WinForm có thể tạo ra các giao diện hiện đại, đẹp như mơ, long lanh hoa mĩ.



Với những ưu điểm trên, WinForm vẫn sống khỏe, sống tốt ở Việt Nam. Tuy nhiên, nếu có ai hỏi mình: Em có nên học và tập trung vào WinForm không? Câu trả lời của mình sẽ là: Học cho biết thì được, chứ **đừng nên tập trung công sức và thời gian vào nó**. Vì sao vậy? Đọc phần sau sẽ rõ.

Đừng nên đầu tư quá nhiều thời gian/công sức vào WinForm

WinForm đã bị Microsoft khai tử, do đó vài năm sau rất **có thể nó cũng sẽ tử ẹo** như VB6. Hiện nay, số lượng các công ty tuyển dụng yêu cầu WinForm cũng không nhiều. Nếu công ty đòi hỏi kiến thức WinForm, có lẽ bạn sẽ phải **bảo trì một (hoặc nhiều) dự án WinForm cũ rích**. Hắn là không bạn nào muốn làm công việc bảo trì nhảm chán qua ngày này tháng nọ phải không?

WPF là đứa đàn em “sinh sau đẻ muộn”, với mục đích kế thừa Win Form. WPF sử dụng ngôn ngữ XAML để làm giao diện nên khá linh hoạt: hỗ trợ animation, tạo user control, ... tốt hơn WinForm nhiều. Microsoft cũng muốn **định hướng cho giới developer sử dụng WPF** để thay thế cho WinForm, có khá nhiều dự án cũng đang migrate từ WinForm sang WPF.

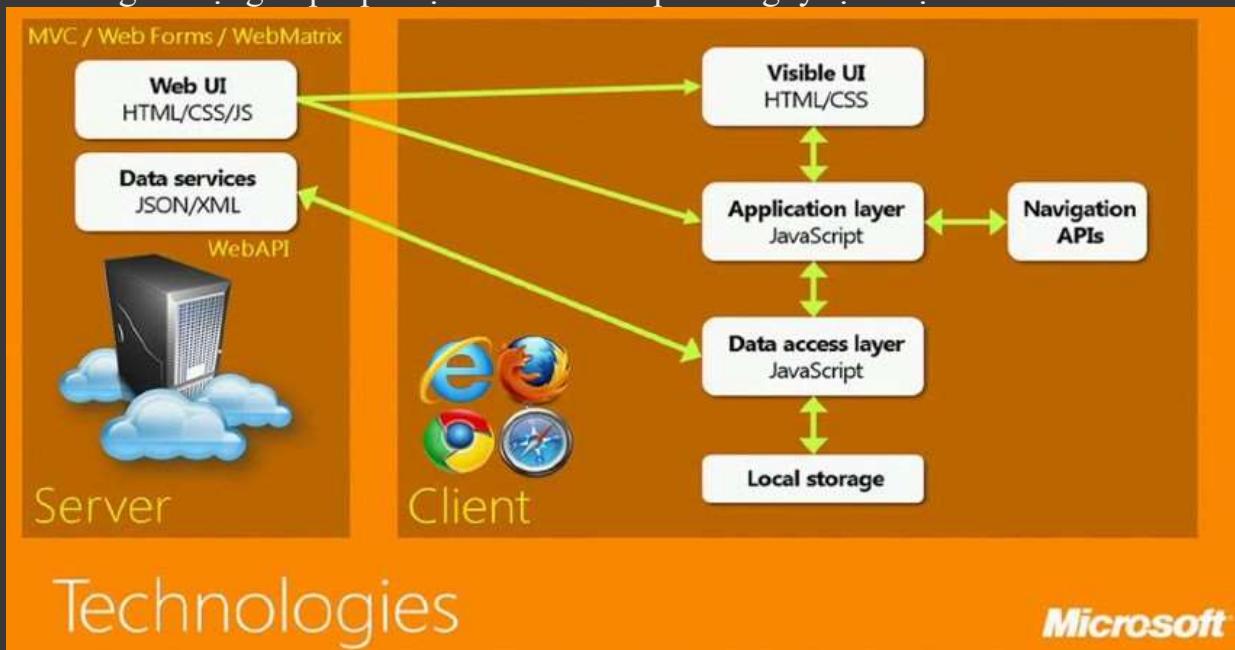


Nhắc lại lời khuyên của mình về WinForm: Có thể thử tiếp xúc với nó, cố gắng nắm vững các khái niệm: Event, Control, Component, ... vì chúng khá có ích. Tuy nhiên, đừng nghiên cứu quá sâu về nó, còn nhiều thứ đáng học hỏi hơn nhiều. Về phần **WPF thì cũng nên thử**, trong WPF có một số khái niệm rất hay: mô hình MVVM, data-binding, ... các khái niệm này khá hữu dụng khi sau này bạn học AngularJS/các javascript framework khác. Kiến thức XAML có được khi học WPF sẽ giúp ích cho bạn khi bạn muốn tập viết app cho [Windows Phone](#).

Còn WebForm thì sao?

Nhắc tới WinForm cũng phải nói đôi chút về WebForm. Từ sau khi ASP.NET MVC ra đời, WebForm có vẻ bị thất sủng. Người thi chê nó chậm, nặng nề (viewstate), kể thi bảo nó khó tích hợp Ajax, khó quản lý HTML. Nói vậy cũng không sai, nhưng lại khá là oan ức cho WebForm. Nó được ra đời để lập trình viên có thể **viết ứng dụng Web một cách nhanh chóng** đơn giản dễ dàng, chỉ cần kéo thả component như WinForm, không cần phải học đủ thứ về Web như Response, Request, Query String, Params,

Nhiều người cho rằng WebForm cũng sẽ chết, và ASP.NET MVC sẽ hoàn toàn thay thế nó. Theo ý kiến cá nhân mình, WebForm **vẫn sẽ sống khỏe, sống tốt**, vì có vô số ứng dụng được xây dựng dựa trên nó, cũng như có vô số lập trình viên có kinh nghiệm WebForm. Một công nghệ sẽ chưa chết chừng nào nó **vẫn còn hữu dụng**. Tuy nhiên, nếu phải chọn giữa WebForm và ASP.NET MVC thì mình khuyên các bạn nên học MVC nhé. Hiện tại các công ty tuyển nhân sự C#.NET đều đòi hỏi kiến thức về MVC. Biết đâu 3-5 năm nữa sẽ có thứ khác hay hơn “đè chết” ASP.NET MVC cũng nên :P. À mà hiện giờ cũng có rồi ấy chứ. Xây dựng ứng dụng dạng SPA (Single-page Application), sử dụng javascript cho front-end và REST API cho back-end cũng là một giải pháp được nhiều developer/công ty lựa chọn.



Một số bài so sánh khá hay về WebForm và ASP.NET MVC:

<http://hoccachlam.com/so-sanh-asp-net-web-form-va-asp-net-mvc/>

<http://tuanitpro.com/tim-hieu-asp-net-mvc-va-su-khac-biet-voi-asp-net-webform>

Trên đây là chỉ suy nghĩ của mình, còn các bạn thì sao? Hãy chia sẻ suy nghĩ/kinh nghiệm của bản thân về WinForm/WebForm trong phần comment nhé.

Review sách: The Healthy Programmer – Giữ sức khỏe để code và ăn chơi phè phỡn

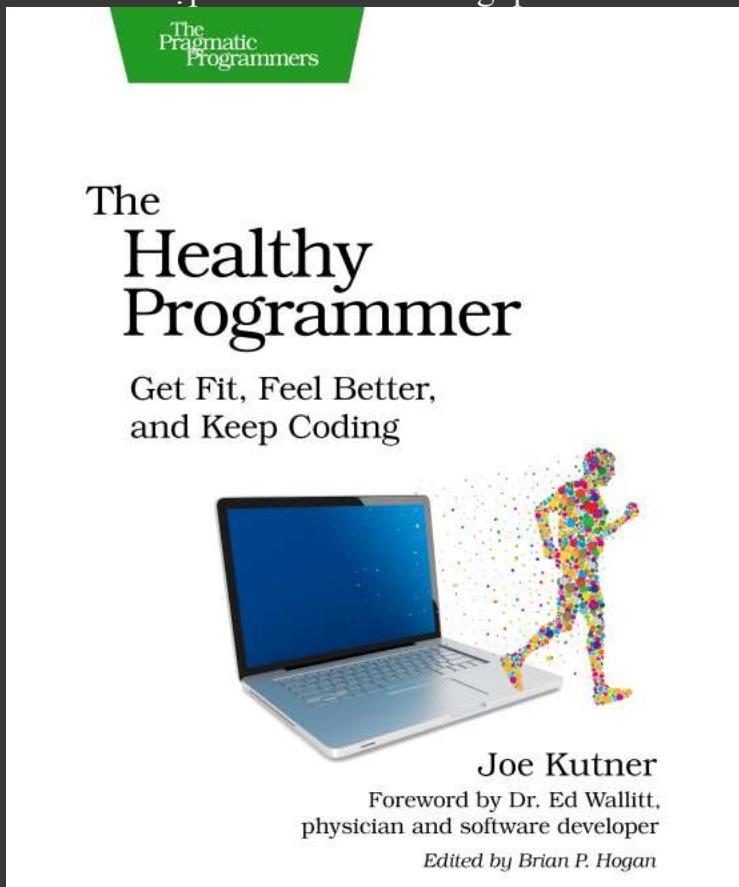
Posted on 01/12/2015 by Phạm Huy Hoàng

Bạn có biết, lập trình là một trong những ngành độc hại?

Bạn có biết, ngoài nhiều sẽ dẫn tới vô số những bệnh trầm trọng và mãn tính?

Bạn có biết, lập trình viên rất dễ mắc các bệnh tim mạch, béo phì, đau khớp, ...?

Hãy đọc The Healthy Programmer – một cuốn sách viết về sức khỏe dành riêng cho developer. Cuốn này mình đọc từ hồi đầu năm khi tình cờ thấy nó trên [it-ebooks](#), nhưng do lười quá nên giờ mới có thời gian review. Như tựa đề, đây là một cuốn sách dành cho dân lập trình, nhưng **nội dung không nói gì về lập trình** mà lại nói đến một vấn đề mà lập trình viên nào cũng quan tâm: sức khỏe.

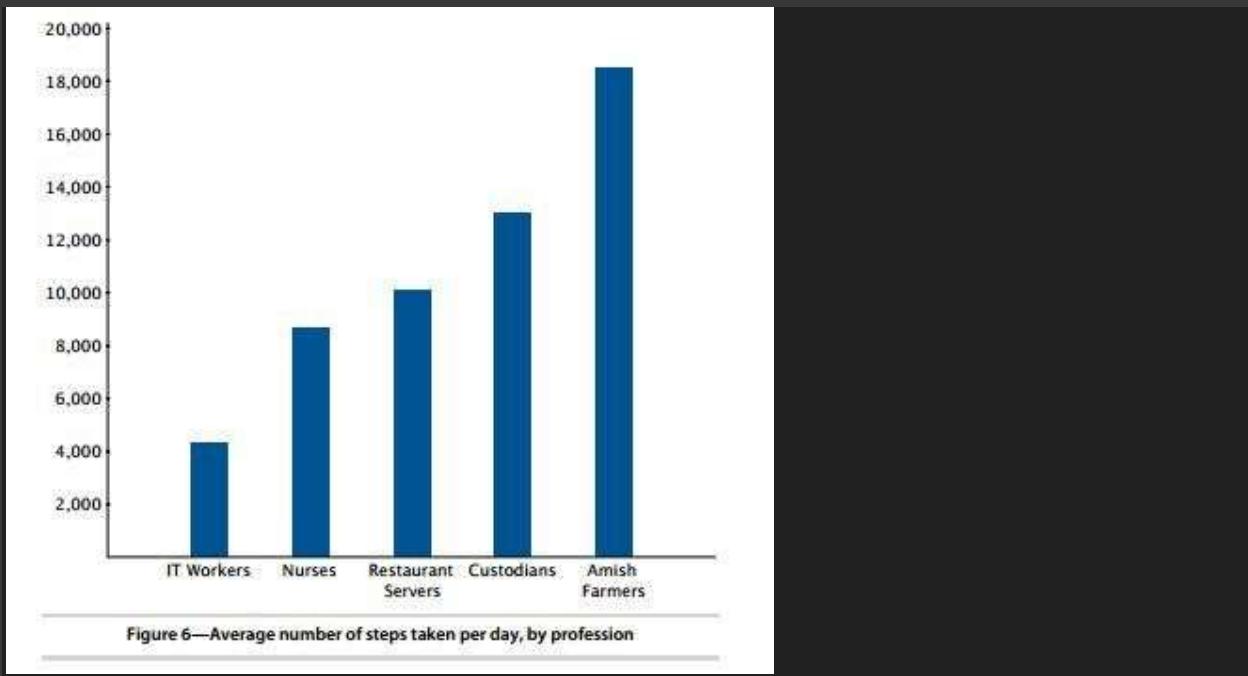


Ở 2 phần của bài viết [Mặt tối của ngành công nghiệp IT](#), có nhiều bạn hỏi mình là tại sao lại không nhắc đến việc “Lập trình viên ngồi nhiều code nhiều nên bụng phệ, tai ương mắt cận, dễ mắc phải đủ thứ bệnh”. Lý do là vì đó là **bệnh chung của dân văn phòng**. Nhân tiện review quyển sách này, mình sẽ giới thiệu một số bệnh, triệu chứng, cách giữ sức khỏe cho dân developer tựi mình. Bạn nào quan tâm đến sức khỏe bản thân, muốn giữ sức để còn đàm đúm chơi bời thì nên đọc.

Giới thiệu tổng quát

Sách mở đầu bằng một câu rất hay: “Lý do quan trọng nhất bạn nên đọc cuốn sách này là sức khỏe của bạn phụ thuộc vào nó, lý do quan trọng thứ nhì là sự nghiệp của bạn cũng phụ thuộc vào nó”. **Phải có sức khỏe tốt thì chúng ta mới theo nghề được lâu dài.**

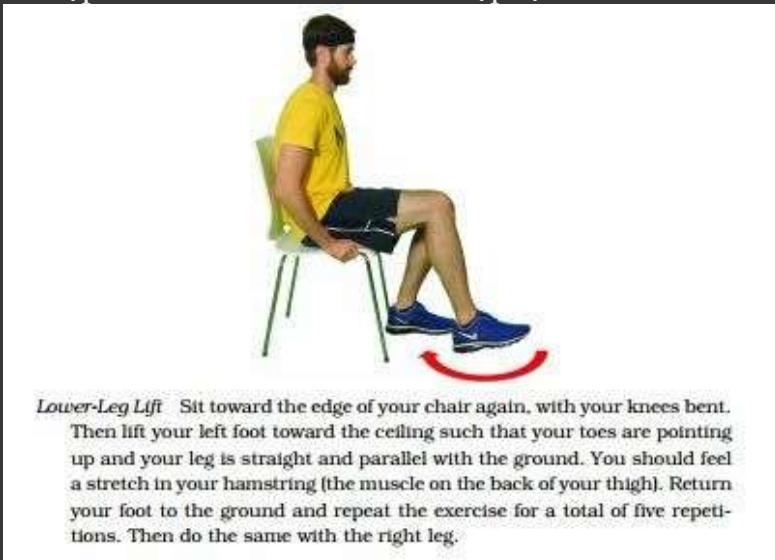
Lối sống của giới developer rất dễ dẫn tới nhiều bệnh tật: Đau mắt, đau đầu, đau lưng, đau khớp tay, tim mạch, béo phì,... Tác giả đã gặp nhiều developer đau lưng mãn tính vì ngồi sai tư thế, hay đau khớp tay vì dùng chuột/bàn phím quá nhiều. Những điều này sẽ gây cản trở không ít thì nhiều cho quá trình làm việc của họ. Vì lẽ đó, cuốn sách này ra đời. (Các ảnh minh họa phía dưới đều cắt từ sách ra nhé).



Developer ngồi nhiều, ít vận động hơn so với các ngành khác nên dễ sinh lâm bệnh

Nội dung các chương trong sách:

- 1. Thay đổi bản thân, xốc lại sức khỏe:** Hướng dẫn ta cách tạo dựng các thói quen tốt cho sức khỏe: Tạo checklist hàng thành, đi bộ mỗi ngày. Việc đi bộ sẽ giảm 20% nguy cơ mắc phải các bệnh thường gặp trong giới văn phòng.
- 2. Tạm biệt chiếc ghế thân thương:** Phân tích tác hại của việc ngồi nhiều. Nếu công việc cho phép, hãy cố gắng ra khỏi chỗ ngồi và đi lại mỗi 30p. Sách cũng có một số bài tập chân cơ bản để ta có thể tập tại chỗ.



- 3. Ăn uống kiểu Agile:** Các bạn developer nữ chắc sẽ thích phần này. Sách hướng dẫn cách xây dựng một khẩu phần ăn hợp lý, cách theo dõi lượng calories thu vào/tiêu thụ trong một ngành để tăng/giảm cân theo ý muốn. Mình thì khoái thịt ghét rau nên làm theo mấy cái hướng dẫn cũng hơi khó. :'(

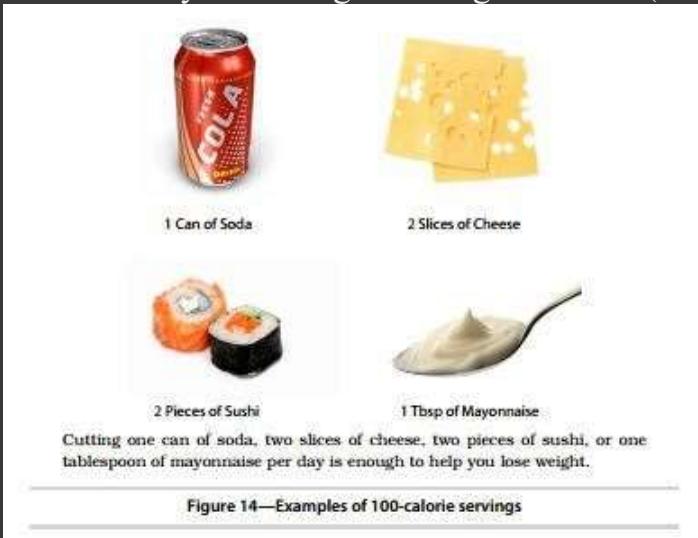


Figure 14—Examples of 100-calorie servings

4. Chóng nhức đầu, đau mắt: Đây cũng là một bệnh mà các developers hay gặp phải. Bạn có thể tìm thấy nguyên nhân, triệu chứng cũng như cách chữa có thể áp dụng ở phần này.

5. Chóng đau lưng, đau tay: Đây là 2 chứng bệnh ảnh hưởng nhiều nhất đến năng suất lao động của developer chúng mình. Triệu chứng của hai bệnh này được mô tả khá rõ ràng. Nếu bạn bắt đầu có các triệu chứng này, hãy thử làm theo các bài tập trong sách hướng dẫn, khá hiệu quả đấy.



Tư thế ngồi đúng chuẩn

6. Ra ngoài và vận động: Ai cũng biết ngồi làm việc trong phòng máy lạnh cả ngày không tốt gì cho sức khỏe. Phần này nói về tác dụng của vitamin D, khuyến khích developer chúng ta nên ra ngoài nhiều. Nội dung cả phần cũng bình thường, có mỗi bài test sức khỏe cuối cùng là hay.

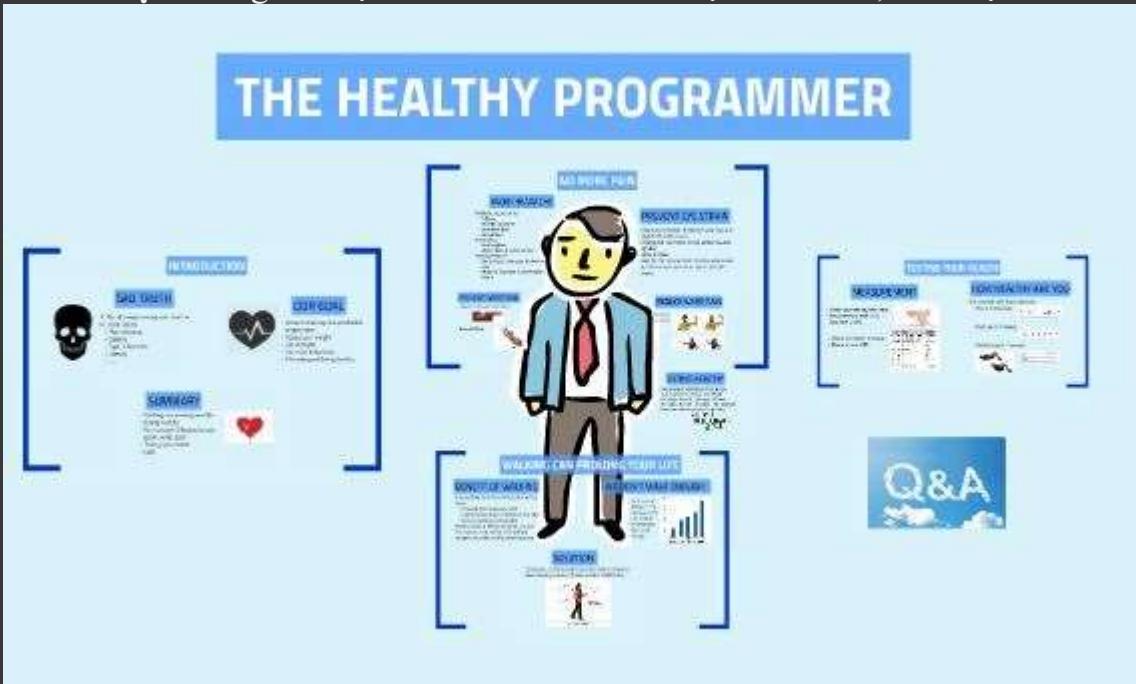
Nhận xét

Công bằng mà nói, cuốn sách đã hoàn thành khá tốt nhiệm vụ của mình. Mặc dù ta có thể tìm các lời khuyên về đau lưng, đau khớp đày rẫy trên google, chúng **không thể sánh bằng các lời khuyên do bác sĩ đưa ra + có chứng cứ khoa học** trong sách.

Sách cũng đã tổng hợp được đầy đủ phần lớn các chứng bệnh mà developer hay gặp phải.

Nếu bạn bắt đầu có những triệu chứng về sức khỏe, hãy đọc sách để tìm cách sửa chữa, tạo thói quen tốt. Nếu sức khỏe bạn vẫn còn tốt, hãy đọc sách để tập thói quen vận động, đồng thời phòng chống những bệnh có thể gặp trong tương lai. Nói chung,

nếu bạn không phải developer nhưng vẫn phải ngồi nhiều, **cuốn sách vãn sê rất có ích với bạn**. Cùng tìm đọc và làm theo để cải thiện sức khỏe, cải thiện bản thân nhé.



Bonus: Mình có làm 1 slide tóm tắt những thứ “hay ho” học được trong sách ở đây, các bạn có thể vào xem

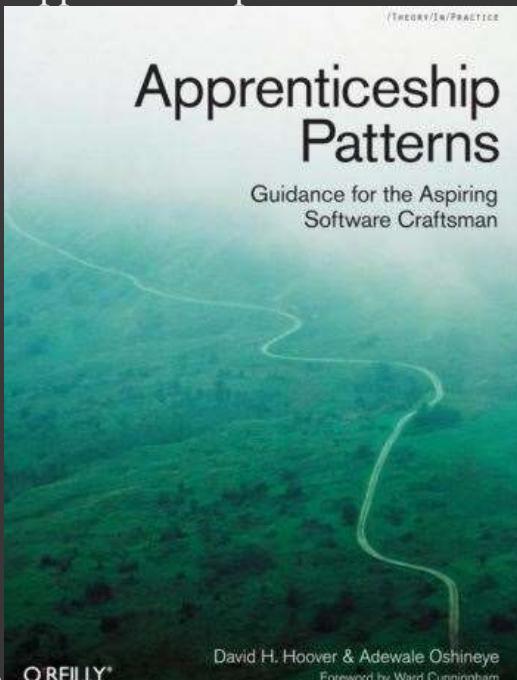
thử: http://prezi.com/ykffcrdliqw1/?utm_campaign=share&utm_medium=copy&rc=ex0share.

Mỗi tháng một cuốn sách – Những sách hay mình đã đọc trong năm 2015 – Phần 2

Posted on 31/12/2015 by Phạm Huy Hoàng

Nối tiếp [phần 1](#), mình sẽ giới thiệu một số sách hay trong giai đoạn tháng 7 tới tháng 12 năm nay.

Tháng 7 – Apprenticeship Patterns – Guidance for the Aspiring Software



Bạn đã đi làm được một thời gian, **cảm thấy chán code**? Bạn mới ra trường, cảm thấy còn **lơ ngơ chưa biết gì**? Bạn nhận thấy con đường trước mắt vẫn còn mù mờ? Đừng lo, hãy đọc cuốn sách này. Sách nâng ngành lập trình lên một tầm cao mới với khái niệm software craftsmanship, người code giỏi cũng như một bậc thầy trong các ngành nghề khác. Sách có vô vàn những mẹo vặt về cách học tập, cách trau dồi kỹ năng, cách giữ lửa đam mê. Bỏ mấy ngày để đọc nó sẽ có ích về lâu dài cho con đường sự nghiệp của bạn nhé.

Có một điều lạ là tuy sách không nổi tiếng lắm (ít review), nhưng hầu như mấy lão senior nước ngoài mình theo dõi đều khuyên đọc cuốn này. Chắc tại vì sách hay, lại được phát hành free, có thể đọc online ở [đây](#).

Một cuốn sách khá hay của tháng này là [Presentation Zen](#) – hướng dẫn cách làm những slide tối giản, cách thực hiện một buổi thuyết trình cuốn hút. Mình đã review cuốn này ở 1 [bài viết trước của blog](#).

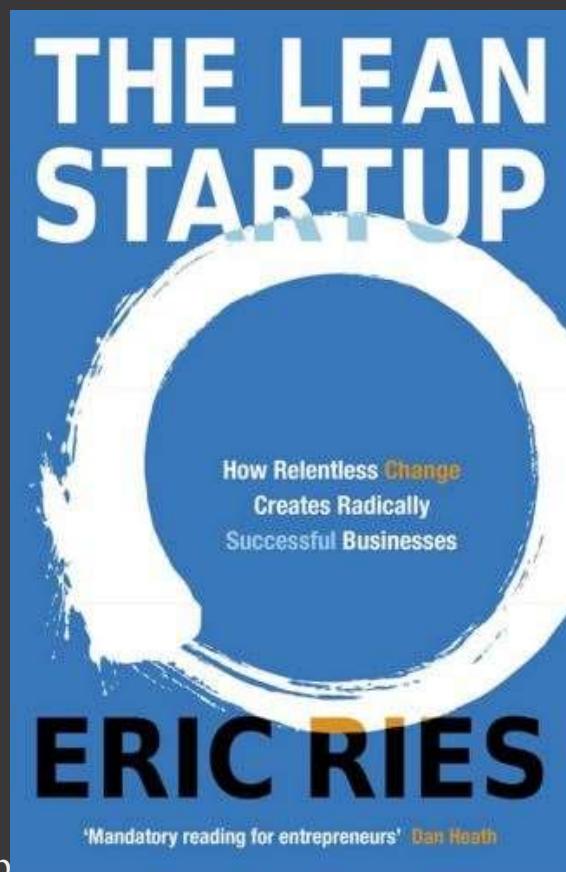


Tháng 8 – Suối nguồn

Đây là cuốn tiểu thuyết duy nhất trong danh sách này. Thấy thẳng bạn cùng công ty review nên mình tò mò đọc thử. Sách dày cuộn, hình như hơn 1200 trang, bản ebook chia ra làm 4 phần, mình ngẫu nhiên phải 4, 5 ngày mới xong.

Suối nguồn là câu chuyện về chàng kiến trúc sư Howard Roast, một người **theo đuổi chân lý đến tận cùng**, không bao giờ thỏa hiệp. Cả câu chuyện là một cuộc đấu tranh giữa chủ nghĩa cá nhân và chủ nghĩa tập thể, giữa cá thể vĩ đại và tập thể tầm thường. Mình đọc mà lâu lâu cũng thấy chính mình trong đó, chắc tại **lập trình và kiến trúc** cũng có vài điểm tương tự.

Nếu rảnh, hãy thử cuốn sách này. Nó sẽ ngốn mất của bạn khá thời gian, nhưng khoảng thời gian đó không hề uổng phí đâu. Lâu lâu đọc danh tác thay vì đọc tin lá cải hay chơi Dota sẽ làm đầu óc ta mở mang nhiều lắm.



Tháng 9 – The Lean Startup

Cuốn sách thứ 2 về startup trong danh sách. Tác giả cũng từng là lập trình viên, từng làm coder chính cho 1 technical startup nên cách viết khá gần gũi. Tuy là sách về startup nhưng mình lại rút ra được nhiều bài học về **phát triển phần mềm** từ cuốn này.

Sai lầm thường thấy của startup và lập trình viên là: Làm ra những sản phẩm/phần mềm **tưởng rất hay nhưng không ai dùng**. Đã bao giờ bạn lập trình một chức năng vì “nó hay”, nhưng người dùng không bao giờ sử dụng chưa? Đó là một sự lãng phí thời gian, tiền bạc, tài nguyên. Với “Lean Startup”, ta **chú trọng vào việc thử-sai (fail early fall fast fail cheaply), không vĩ đoán**. Thay vì dự đoán rằng người dùng sẽ thích sản phẩm, chức năng nào đó, hãy thử tạo ra nó; rồi kiểm tra phản ứng của người dùng, sau đó quyết định nên dừng lại hay tiếp tục.

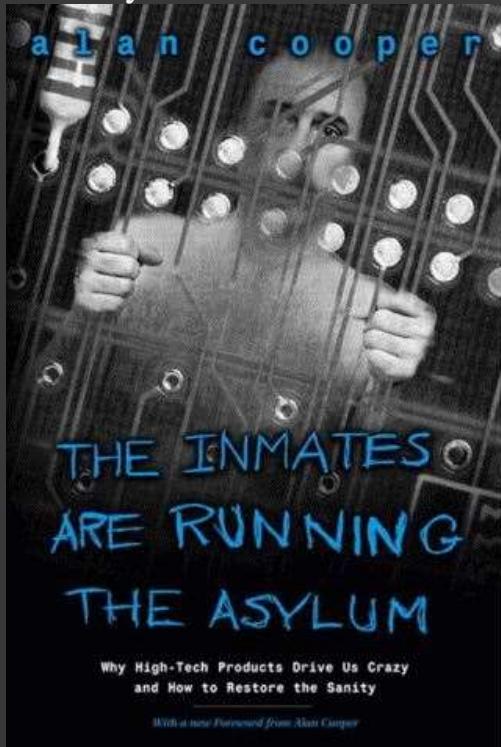
Một cuốn sách khác hay của tháng là *Breakthrough rapid reading*, dạy các bạn cách tăng tốc độ đọc sách, cách tiếp thu và nhớ kiến thức từ sách. Mình áp dụng và thấy tốc độ đọc cũng tăng đáng kể, bộ 1Q84 khá dài của Haruki chỉ đọc mất 3-4 ngày là xong.

Tháng 10 – Kẻ thành công phải biết lắng nghe



Mình mua cuốn này để đọc khi lên máy bay sang UK [du học](#). Sách hướng dẫn ta cách lắng nghe, thấu hiểu người khác. Giữa cái xã hội toàn những thanh niên chém gió, nói giỏi hơn làm, **tìm một người biết cách lắng nghe thực sự rất khó**. Đọc sách, bạn sẽ hiểu được cách não bộ hoạt động, cách làm nguôi giận ngừng giận dữ, cách đồng cảm với người khác, cách dù dỗ người khác thổ lộ với mình... Đối với mình, đây là một cuốn sách khá hay về kỹ năng sống, chỉ đứng sau [Đắc Nhân Tâm](#).

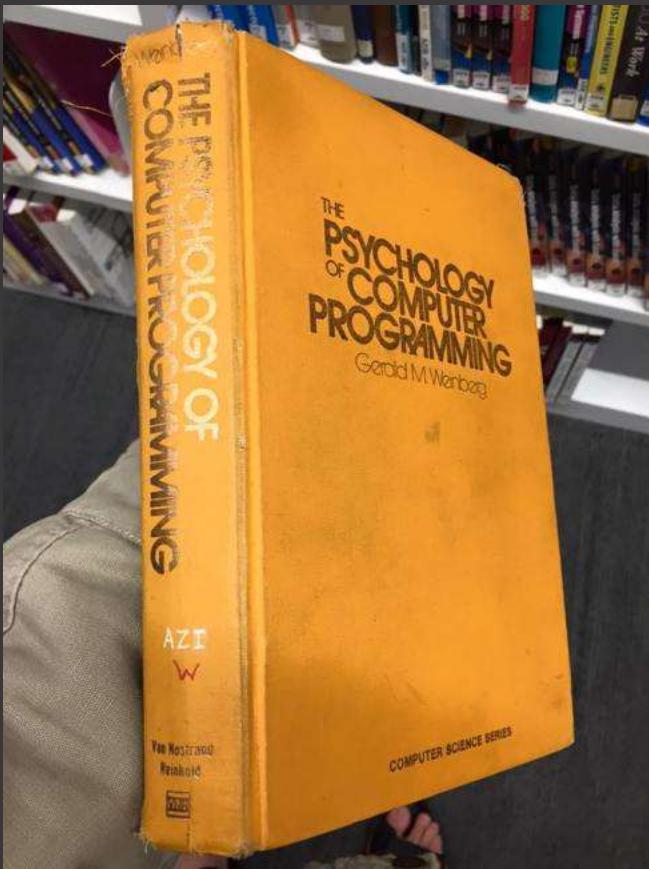
Tháng 11 – The Inmates are Running the Asylum – Why High-Tech Products Drive Us Crazy and How to Restore the



Sanity

Mình đã từng nhắc đến cuốn này ở bài: [Lập trình viên trình cao nên đọc sách gì?](#). Tác giả đã nêu lên thực trạng của các phần mềm/phần cứng hiện nay: Chúng quá phức tạp, quá khó sử dụng, gây khó khăn cho người dùng. Ông đưa ra một số ví dụ về các thiết bị được thiết kế cẩu thả: máy ảnh số, máy ATM, điều khiển từ xa, ... Sách đề cao tầm quan trọng của trải nghiệm người dùng (UX – User Experience), UX nên được thiết kế trước tiên trước khi thiết kế phần mềm. Đọc xong cuốn này, bạn sẽ ngộ ra được rất nhiều “chân lý”.

Tháng 12 – The Psychology of Computer Programming



Do cơ duyên run rủi cho mình bắt gặp cuốn này trong kệ sách của thư viện trường. Sách ra đời năm 1971, tính ra chắc tuổi đời cũng già hơn phần đông các bạn ở đây. Bìa đã hơi long tí chút, nhưng do bảo quản kĩ nên bên trong sách còn khá mới.

Tại sao một cuốn sách tuổi đời hơn 40 năm vẫn nằm trong danh mục **sách-khuyên-đọc** cho quản lý và lập trình viên? Bởi vì “trong cái thế giới mà công nghệ thay đổi từng ngày, vẫn có những thứ kiến thức không bao giờ lạc hậu”. Đây là **cuốn sách đầu tiên viết về nhân tố con người** trong ngành lập trình.

Đọc sách, ta sẽ có một cái nhìn mới phương pháp làm việc nhóm, suy nghĩ của lập trình viên, cách khen thưởng và huấn luyện nhung viên. Tuy đã qua 30 năm, công nghệ thay đổi, ngôn ngữ lập trình thay đổi nhưng những thứ này vẫn còn y nguyên. Có đôi chỗ hơi khó chịu khi đọc là sách có dùng một số khái niệm + ngôn ngữ lập trình khá cũ nên nhiều đoạn về technical mình không hiểu mấy.

Bonus: Danh sách các sách mình đã đọc từ tháng 7 – tháng 12, sách **in đậm** là sách hay nhất tháng, *in nghiêng* là sách hay – đáng đọc nhé.

Tháng 7

- *Trai súc vật*
- **Apprenticeship Patterns – Guidance for the Aspiring Software Craftsman**
- On Writing Well – The Classic Guide to Writing Nonfiction
- Kinh tế học và sex
- *Presentation Zen*
- *Điểm bùng phát – The tipping point*
- Thriving on Less
- Mật ngọt chết mèo
- *What The Dog Saw: And Other Adventures*

Tháng 8

- *David and Goliath*
- Vui chơi để kiếm sống
- *Slide:ology: The Art and Science of Creating Great Presentation*
- Nếu gặp người ấy cho tôi gửi lời chào
- Sư im lặng của bầy cừu
- **Suối nguồn**
- *Microsoft dotNET – Architecting Applications for the Enterprise*
- *Microsoft dotNET – Architecting Applications for the Enterprise, 2nd Edition*
- *The \$100 Startup*

Tháng 9

- Fooled by randomness
- *NoSQL Distilled*
- Âm mưu ngày tận thế
- Giết con chim nhại (dở)
- **The Lean Startup**
- *1Q84*
- *Breakthrough rapid reading* (Khá hay)
- 20 tuổi trở thành người biết nói, giỏi làm

Tháng 10

- *Trên đường băng*
- Yêu người ngóng núi
- *After Dark – Haruki Murakami*
- Đi đâu cũng nhớ Sài Gòn và ... Em

- *Economics for Dummies*
- **Kẻ thành công phải biết lắng nghe**
- *The Naked Presenter*
- How to succeed in written work and study
- Doing Essays and Assignments: Essential Tips for Students
- *Data Mining for Dummies*
- MATLAB – A practical introduction to programming and problem solving
- Just write – An easy-to-use guide at writing in university
- Writing Essays at University A Guide For Students, By Students
- Inside track to successful academic writing

Tháng 11

- Data Mining, Concepts and Techniques, Third Edition (partly)
- The Black Swan (về sau dài dòng nhảm)
- *Anti Patterns: Refactoring Software, Architectures, and Project in Crisis*
- Purple Cow: Transform Your Business by Being Remarkable.
- It's Not How Good You Are, It's How Good You Want to Be
- **The Inmates are Running the Asylum – Why High-Tech Products Drive Us Crazy and How to Restore the Sanity**
- Manga! Manga! The World of Japanese Comics

Tháng 12

- **The Psychology of Computer Programming**
- Search me – The surprising success of Google
- Thỏ bảy màu – Timeline của tui có gì
- *Atlas Shrugged*
- Chuyện con ốc sên muốn biết vì sao nó chậm chạp
- *The Book Thief*
- *Dreaming in Code*
- *Remote – Office not required*
- *Năm Cam – Canh bạc cuối cùng*

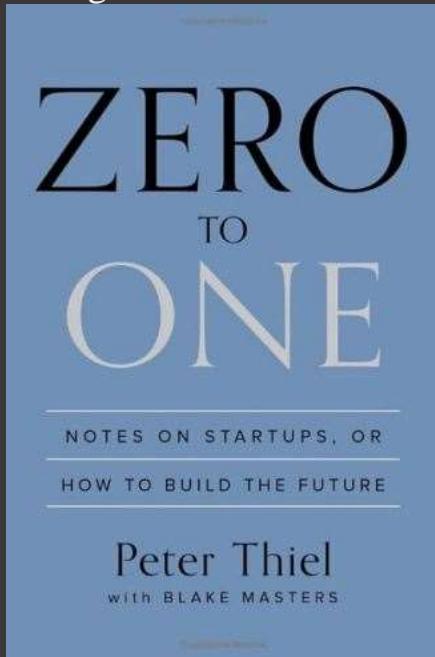
Mỗi tháng một cuốn sách – Những sách hay mình đã đọc trong năm 2015 – Phần 1

Posted on 29/12/2015 by Phạm Huy Hoàng

Từ khi ra trường, mình vẫn luôn giữ thói quen đọc sách, một phần là để giải trí, một phần là để học hỏi cái mới. Dù cho không học hỏi, áp dụng được gì thì cảm giác cầm cuốn sách (hoặc iPad) nghiền ngẫm từng chữ cũng khá thú vị. Để khuyến khích thói quen đọc sách, cũng như chia sẻ sở thích với một số bạn, bài viết này sẽ là review tổng hợp ngắn những cuốn sách hay nhất mình đã đọc trong năm vừa rồi.

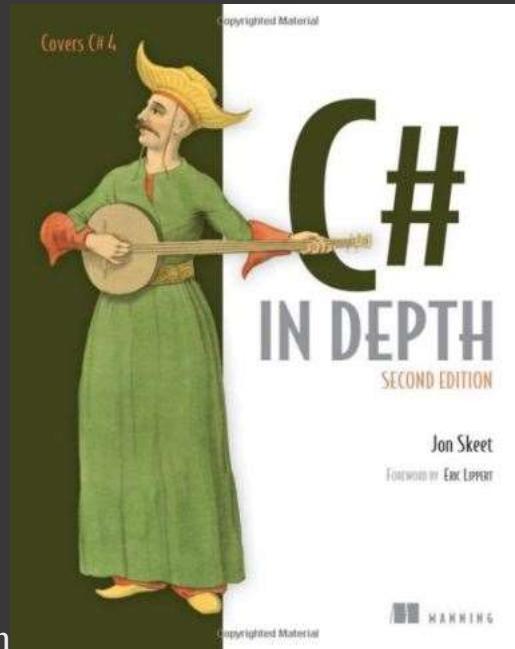
Do mình đọc khá nhiều thể loại: technical, kỹ năng sống, kinh tế, marketing, tiểu thuyết, tản văn, ... nên những sách giới thiệu sẽ không theo một chủ đề cố định nào cả. Mỗi tháng mình đọc khoảng 6,7 cuốn sách. 12 cuốn sách dưới đây là 12 cuốn sách hay nhất mà mình đọc được mỗi tháng; có tháng mình đọc được khá nhiều cuốn hay nhưng chỉ chọn được 1 cuốn hay nhất để giới thiệu. Các bạn có thể xem toàn bộ danh sách ở cuối bài.

Tháng 1 – Zero to One: Notes on Startups, or How to Build the Future



Đây là một cuốn sách khá hay về startup. Tác giả chỉ ra những mindset mà người làm startup nên có, những khó khăn, những điều cần phải lường trước. Ông cũng đưa ra 7 câu hỏi mà người start-up phải trả lời: Bạn đã xác định được 1 cơ hội mà nhiều người không thấy chưa? Ngoài việc tạo ra sản phẩm, bạn đã nghĩ ra cách đưa nó đến tay

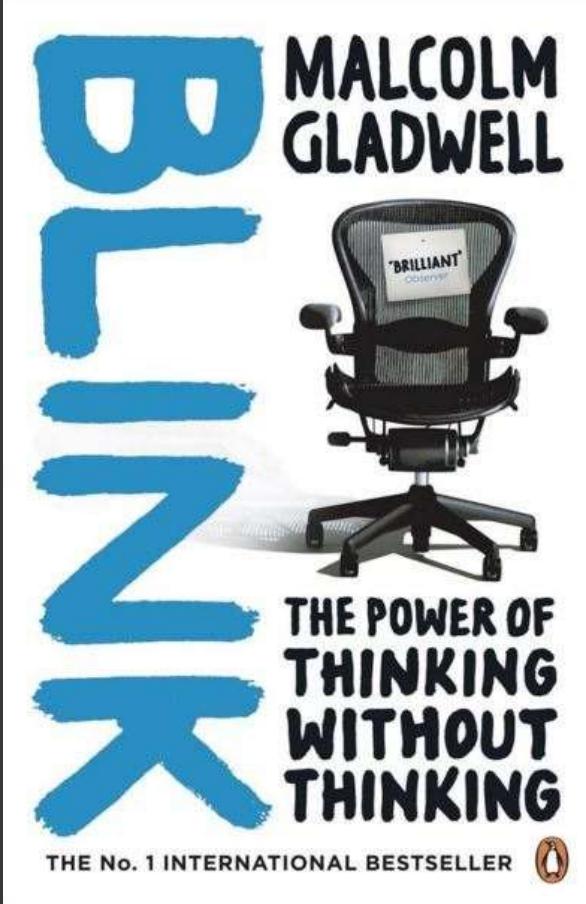
người dùng chưa?... Mình chưa có ý định làm startup, nhưng vẫn thấy những ý tưởng trong sách khá hay và đáng học hỏi.



Tháng 2 – C# in Depth 3rd Edition

Một cuốn sách thuần túy technical về ngôn ngữ C#. Nhờ đọc cuốn này, mình nghiên ngẫm ra được nhiều khái niệm mà trước giờ vẫn hơi mơ hồ như: callback, delegate, lambda expression, generic, LINQ, ... và viết được series C# hay ho cho blog này. Cuốn này khá hữu dụng cho cả junior lẫn senior developer.

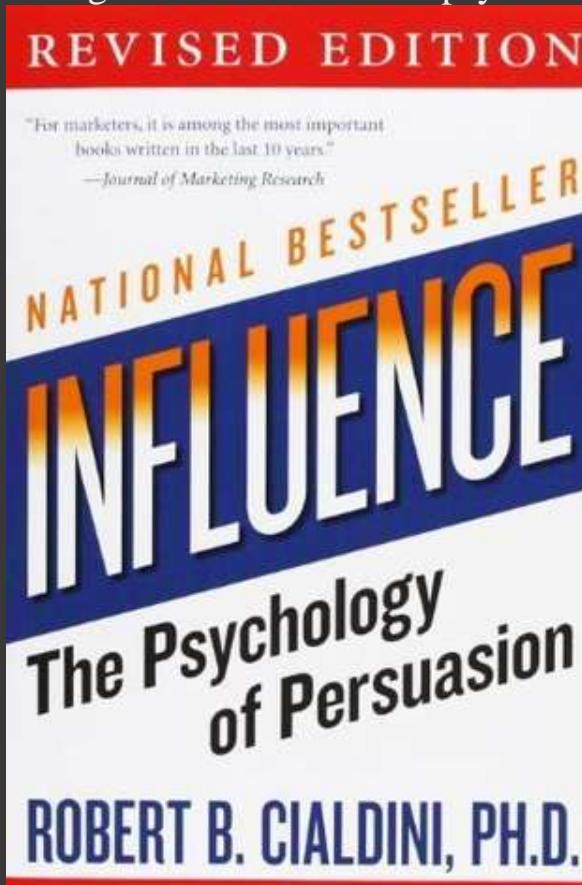
Tháng 3 – Blink: The Power of Thinking Without Thinking



Tác giả của cuốn sách là Gladwell Malcoml, một cây viết già đời của tờ The New Yorker, với những lối suy nghĩ, cái nhìn lạ. Cuốn sách này nói về cách ta sử dụng tiềm thức – suy nghĩ mà không cần suy nghĩ, đưa ra quyết định “trong chớp mắt”. Cách viết của Gladwell rất cuốn hút và hấp dẫn, đi kèm với những cái nhìn lạ là những câu chuyện thú vị.

Cuốn này đã xuất bản ở Việt Nam với tựa là Trong Chớp Mắt. Những cuốn còn lại của tác giả cũng rất hay: David and Goliath, The Outlier, The Tipping Point, ... mình thành fan của Gladwell nên nuốt nốt mấy cuốn đầy luôn.

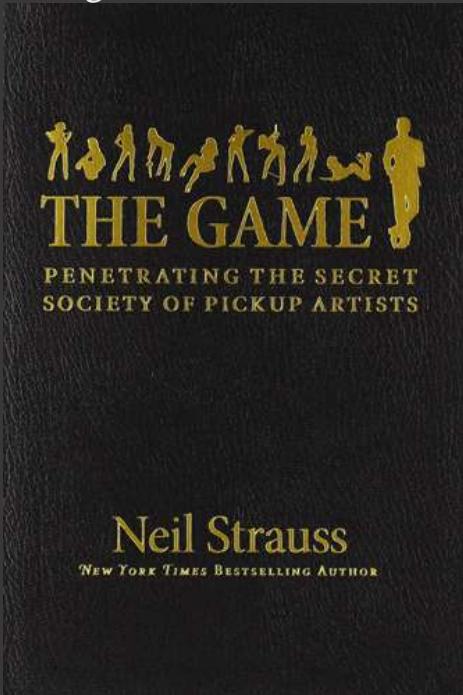
Tháng 4 – Influence – The psychology of Persuasion



Mình được một anh bạn đồng nghiệp ASWIG (cũng ham đọc sách) giới thiệu cuốn này. Ban đầu, mình tưởng nó chỉ là sách marketing bình thường, nhưng càng đọc mình càng bị thu hút. Cuốn sách hướng dẫn cách thuyết phục, thay đổi người khác. Trong cuốn sách, bạn sẽ học được **6 nguyên lý cơ bản để thuyết phục** người khác. Đây là những nguyên lý mà tự quảng cáo, marketing áp dụng hàng ngày, đôi khi chúng ta cũng vô tình (hoặc cố ý) sử dụng mà không hay biết. VD như nguyên lý thứ sáu, **scarity – cái gì hiếm thì sẽ quý**, do đó các siêu thị hay chơi trò “Số lượng có hạn, mua ngay kẻo hết”.

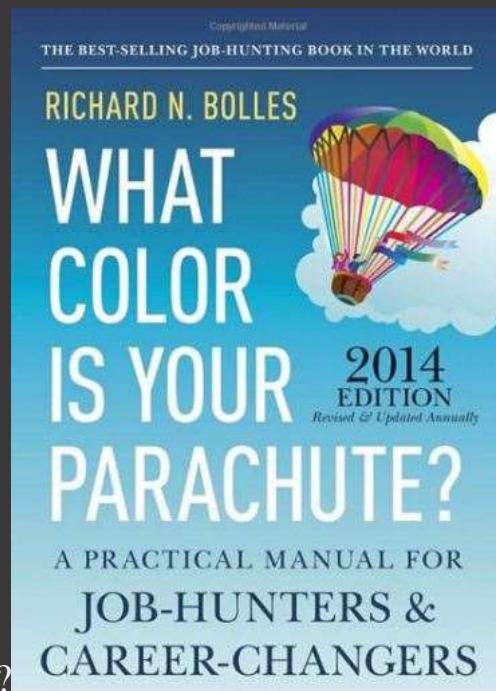
Đây là một cuốn sách có thể thay đổi cách nhìn của bạn, cũng như dạy bạn thêm về cách thuyết phục bạn bè, đồng nghiệp, cấp trên, hãy tìm ebook hoặc mua đọc thử nhé.

Tháng 5 –The Game: Penetrating the Secret Society of Pickup Artists



Đây là sách thể loại nửa tiểu thuyết, nửa hồi ký, là một câu chuyện có thật của anh chàng Neil Strauss (Bạn nào hay đọc vozforum chắc cũng biết thể loại tương tự). Trong 1 lần tìm đè tài viết bài gửi cho tòa soạn, Neil đã gặp Mystery, một gã trùm “chǎn rau” thứ thiệt. Từ một thanh niên “đụt”, dưới sự dạy dỗ của Mystery, anh học được những mánh khép tiếp cận phụ nữ, những chiêu trò lấy lòng và **đưa họ lên giường**. Một thế giới nơi phụ nữ chỉ là *con mồi*, là *phần thưởng*, nơi tình yêu, rung động chỉ là những chiêu trò tâm lý...

Sách viết khá hay, khá thật và quan trọng là **kiến thức trong sách áp dụng được vào thực tế**. Các bạn nam – nhất là dân lập trình nên đọc cuốn này, nó sẽ hướng dẫn bạn cách tiếp cận phái nữ, cách đánh bóng bản thân, cách nhắn tin cho gái, phương pháp kéo – thả, ... Nếu ham học hỏi bạn nên đọc thêm cuốn *The Mystery Method: How to Get Beautiful Women into Bed* do chính nhân vật Mystery người thật việc thật chắp bút. Học để kiểm gấu thì được chứ đừng dùng để chǎn rau nhé !!!



Tháng 6 – What color is your parachute?

Cuốn này mình tình cờ được anh team leader giới thiệu. Đây là một cuốn sách rất hay về việc xác định kỹ năng, hướng đi của bản thân. Đọc sách, bạn sẽ biết mình là ai, mình muôn gì, mình có thể làm gì, để xác định con đường phát triển sự nghiệp của chính mình. Sách cũng hướng dẫn một số kênh tìm việc thường thấy, cách viết một bức thư xin việc, cách chuẩn bị phỏng vấn.

Nói không ngoa, đây là một trong những cuốn sách hay nhất về việc làm và nghề nghiệp; đó cũng là lý do nó liên tục được cập nhật phiên bản mới mỗi năm từ 1970 tới giờ.

Ở phần 2, mình sẽ giới thiệu 6 cuốn sách còn lại, các bạn đón đọc nhé.

Bonus: Đây là list các sách mình đã đọc từ tháng 1 tới tháng 6, cuốn nào **in đậm** là hay nhất tháng, *in nghiêng* là sách hay – đáng đọc nhé.

Tháng 1

- Cuốn theo chiều gió
- Tuyển tập truyện ngắn Haruki Murakami
- *Superhuman by Habit*
- **Zero to One: Notes on Startups, or How to Build the Future**
- Lắng nghe gió hát – Haruki Murakami

- Quốc gia khởi nghiệp
- The Magic of Thinking Big
- *The Pragmatic Programmer, From Journeyman To Master*
- *The Passionate Programmer Creating a Remarkable Career*
- The Road Less Traveled

Tháng 2

- **C# in Depth 3rd Edition**
- Yêu đi thôi, muộn lăm rồi
- Nghĩ giàu làm giàu (dở)
- Hackers and Painters
- 5 cm trên giây
- *Nhà tự nhiên kinh tế – Tại sao kinh tế học có thể lý giải mọi điều*
- Chuyện con mèo dạy con hải âu bay

Tháng 3

- Innumeracy: Mathematical Illiteracy and Its Consequences
- **Blink: The Power of Thinking Without Thinking**
- Năm tháng là đóa lưỡng sinh hoa
- Cô gái văn chương và người hành hương than khóc
- Tam sinh tam thế – Thập lý đào hoa
- *Outliers – The story of success*
- Không bao giờ là thất bại – Tất cả là thử thách
- *Một đời thương thuyết*
- Đong tấm lòng
- Bảy bước tới mùa hè

Tháng 4

- The Mastery of Love: A Practical Guide to the Art of Relationship: A Toltec Wisdom Book
- **Influence – The psychology of Persuasion**
- Hoàng tử bé
- *The Full Facts Book of Cold Reading: A Comprehensive Guide to the Most Persuasive Psychological Manipulation Technique in the World*
- *The Four Agreements: A Practical Guide to Personal Freedom*
- Ngôn ngữ của chúa

- *What every BODY is saying*
- Ame & Yuki – Những đứa con của sói
- Quái vật giữa loài người
- 10 nghịch lý cuộc sống

Tháng 5

- **The Game: Penetrating the Secret Society of Pickup Artists**
- Tớ là mèo Pusheen
- *The Mystery Method: How to Get Beautiful Women into Bed*
- *Steve McConnell – Rapid Development – Taming Wild Software Schedules*
- Cuộc đời của Pi
- Programming Pearls
- Nhà giả kim

Tháng 6

- Sài Gòn tả pín lù
- *All marketers are liars*
- *Cracking the Coding Interview – 150 Programming Interview Questions and Solutions*
- Facts and Fallacies of Software Engineering
- 58++ bài học vỡ lòng để trở thành No.1
- Cái cười của thánh nhân
- *Code – The Hidden Language of Computer Hardware and Software*
- Người giỏi không bởi học nhiều
- *The Elements of Style*
- **What color is your parachute?**

Những mánh khóc “không bao giờ tiết lộ” của các lập trình viên vĩ đại

Posted on 24/12/2015 by Phạm Huy Hoàng

Mánh khóc thứ nhất: Không bao giờ truyền dạy các “mánh khóc” này cho người khác. Hết bài.

Đùa đà, các bạn đừng gạch đá mình tội nghiệp, mình buồn. Chẳng là đạo này mình chán [stackoverflow](#), chuyển qua [quora](#) nghịch ngợm đôi chút. Đây cũng là một trang web hỏi đáp tương tự như stackoverflow, nhưng phạm vi rộng hơn rất nhiều, bao gồm toàn bộ mọi lĩnh vực đời sống.

Một điểm đặc biệt nữa là nó cho phép hỏi những câu chung chung hoặc “nhảm nhí”, do đó có rất nhiều câu hỏi – trả lời thú vị và “bá đạo” như: Mac Zuckerberg có giỏi PHP hay không? Tại sao người đời lại ghét sản phẩm của Apple? Làm sao nghe lén điện thoại bạn gái?... đủ thứ trên trời dưới đất =))).

Mình khuyên các bạn nên bỏ đi ít thời gian cho facebook, rảnh rỗi thì lên đây xem các câu hỏi về Computer Science/Computer Programming, sẽ học được nhiều điều thú vị lắm, giải trí cũng tốt nữa. Bài viết này là tổng hợp và chọn lọc những ý tưởng, câu trả lời hay của câu hỏi: [What are the best-kept secrets of great programmers?](#) – Những mánh khóc “không bao giờ tiết lộ” của các lập trình viên vĩ đại.



Mánh khóc code và test

- Trong đa phần các trường hợp, sử dụng inheritance (kế thừa) là một **design TÊ**, làm cho code khó test và khó bảo trì. Hãy chuyển qua composition (sở hữu) và

kết hợp với interface. (Có thể đọc thêm về *prefer composition over inheritance*).

- Đừng sử dụng interface cho tới khi bạn hoàn toàn rõ ràng về domain của chương trình. (Mỗi khi cần thêm 1 function, bạn sẽ phải thêm nó vào interface và implement của interface đó, gấp đôi công sức).
- Bảo mật/mã hóa rất khó. Đừng tự làm Mà hãy tái sử dụng (sử dụng thư viện, thuật toán có sẵn v...v), trừ khi bạn biết rõ mình đang làm gì.
- **Có vô vàn nguyên nhân làm crash một chương trình:** deploy sai cách, input bị lỗi, người dùng dùng sai cách, quá tải ... Chuẩn bị sẵn sàng cho những điều đó: Ghi log những exception gặp phải, deploy thử lên server test, đặt giới hạn cho bộ nhớ...
- Kết nối mạng (HTTP, socket) rất dễ xảy ra vấn đề. Luôn nhớ đặt timeout cho các kết nối này, sử dụng thư viện để wrap chúng, retry nếu kết nối có vấn đề.
- Mỗi dòng code thêm vào sẽ làm chương trình phức tạp thêm một chút, tăng khả năng có bug. Bỏ bớt code là cách hay nhất để giảm bớt số lượng bug =))).
- Validate những thứ người dùng nhập vào, vừa đảm bảo tính bảo mật, lại hạn chế được bug.
- Tái sử dụng code chưa chắc đã khiến code của bạn dễ bảo trì hơn. Tái sử dụng code giữa 2 domain khác nhau có thể làm chúng “dính chặt” với nhau hơn.
- Chỉ test những thứ cần test, test ít thì dễ sót bug, test nhiều thì sẽ mất thời gian và tốn công update test case mỗi khi đổi requirement.
- Mỗi khi commit code, hãy giữ số lượng code nhỏ, code chạy được, viết message rõ ràng bao gồm *thứ bạn đã làm và lý do bạn làm thứ đó*.
- Với *kiến trúc tốt*, bạn vẫn có thể viết *code lô*. Tuy nhiên, với kiến trúc tốt, bạn có thể dễ dàng nâng cấp, thay thế phần code đó. Tập trung xây dựng kiến trúc tốt, ít móc nối trước, về sau sẽ dễ thay đổi.
- Code để lâu cũng rất dễ hư hỏng, do đó cần được refactor thường xuyên. Tuy nhiên cần tránh refactor code quá độ.



Mánh khóe làm việc

- Rất khó để ước đoán thời gian cần làm để hoàn thành một module/dự án, đó là lý do người ta dùng **Scrum**.
- Viết code để cho chính mình và người khác đọc. Thêm comment để giải thích “Vì sao”, thêm comment ở những nơi mà bạn nghĩ **1 năm sau bạn đọc code sẽ không hiểu gì**.
- Hiểu rõ thư viện/framework mà mình sử dụng, *đừng có gắng viết lại từ đầu*những thứ người khác đã tốn công viết rồi.
- Cài đặt để việc build một project diễn ra nhanh chóng tiện lợi nhất có thể. Hãy chắc chắn bạn có thể build bằng command line, sẽ rất có ích (Có thể kích hoạt build từ xa, hoặc đưa project lên CI chẳng hạn).
- Hiểu rõ những tool bạn sử dụng (IDE, source control, build tool, Photoshop). Có gắng tìm hiểu và làm quen với việc dùng các hotkey, hạn chế dùng chuột. Bạn sẽ làm việc nhanh hơn và “pro” hơn.
- Ngồi lâu rất có hại. Hãy tập một số thói quen để đảm bảo sức khỏe khi làm việc: Không ngồi nhiều, lâu lâu cho mắt nghỉ ngơi, sắp xếp bàn làm việc, bàn phím, chuột sao cho làm việc thoải mái...
- Đừng áp dụng lung tung các framework/process/pattern vào dự án để “thể hiện”. Không phải lúc nào Test-Driven Development cũng tốt, không phải lúc

nào cũng nên áp dụng [DI/IoC](#).



Mánh khóe phát triển bản thân

- Vọc code của các ứng dụng, framework Open Source là cách nhanh nhất để học hỏi và “lên trình”.
- Code review là **một trong những cách hay nhất giúp bạn tiến bộ**, có người đánh giá code của bạn, giúp bạn phân biệt code giỏi và dở, tránh những lỗi lầm cơ bản (Ở Việt Nam mình thấy việc code review này làm khá qua loa, khá chán).
- Học một ngôn ngữ mới sẽ giúp bạn *hiểu những khái niệm mới*, có cái nhìn mới, cách suy nghĩ sẽ linh hoạt hơn. (Thử chuyển từ C#/Java sang scripting language như python/javascript bạn sẽ thấy một chân trời mới).
- Học một ngôn ngữ hướng đối tượng là chuyện dễ. Biết cách *thiết kế hệ thống theo hướng đối tượng* là chuyện khó. Hãy tìm hiểu các [nguyên lý SOLID](#) và một số Design Pattern, chúng sẽ nâng cao hiểu biết của bạn về thiết kế hướng đối tượng.
- Luôn giữ tinh thần học hỏi, nhưng **đừng chạy theo công nghệ mới**. Đừng chọn một công nghệ cho một dự án chỉ vì nó hot/mới/hay.

Lỗ hổng bảo mật khủng khiếp của Lotte Cinema (Lưu trữ mật khẩu người dùng – Tưởng dẽ mà không đơn giản)

Posted on 22/12/2015 by Pham Huy Hoàng

Đăng nhập là một chức năng đơn giản nhất mà hơn 90% các [trang web](#) cần phải có. Tuy nhiên, đôi khi ta lại không được hướng dẫn cách thực hiện chức năng “Đăng nhập” một cách đúng đắn, bài bản, dẫn đến những lỗi dở khóc dở cười, hoặc những **lỗ hổng bảo mật khủng khiếp**. Đến cả Lotte Cinema, một trang web được khá nhiều người dùng còn mắc lỗi sơ đăng này.



Đăng nhập hả? Chỉ cần một bảng User, hai cột Username và Password là xong

Kẻ cũng buồn cười. Ngày xưa khi đi học, mình được hướng dẫn cách làm chức năng đăng nhập như thế này.

1. Người dùng nhập tên tài khoản (email) và mật khẩu.
2. So sánh tên tài khoản và mật khẩu với thông tin trong database.
3. Nếu đúng, cho người dùng đăng nhập, lưu thông tin vào session hoặc cookies.

Bước 1 và 3 không có gì đáng bàn, nhưng bước 2 mới là điều đáng nói. Đa phần tụi mình đều **lưu trực tiếp tên tài khoản và mật khẩu** vào database, sau đó đem ra so sánh.

```
1     public void Register(string username, string password)  
2     {
```

```
3     Database.SaveUser(username, password);
4 }
5
6     public bool Login(string username, string password)
7 {
8     string pw = Database.GetPasswordByUsername(username);
9     return pw == password;
10 }
```

Đây là cách **củ chuối nhất và ngu nhất**. Database là một trong những nơi hay bị tấn công, dễ làm thất thoát dữ liệu. Trong quá khứ, lỗi SQL Injection từng làm thất thoát hàng triệu thông tin khách hàng và thông tin credit card. Chưa tính đến chuyện hacker bên ngoài, nhiều khi thằng Database Admin hưng lên, nó có thể mò được mật khẩu của khách hàng, lớn chuyện chưa?

Cách lưu trữ mật khẩu đúng phải là **làm sao để chỉ người dùng mới biết được mật khẩu của họ**. Làm sao ư? Hãy xem phần dưới nhé.



Vậy mã hóa là được chứ gì, lăm tróc!!

Ừ, cách giải quyết cũng khá đơn giản. Bạn có thể dùng hàm hash để mã hóa mật khẩu như sau:

1. Sử dụng hàm hash (hàm băm) để mã hóa mật khẩu của người dùng.
2. Lưu trữ mật khẩu này dưới database.
3. Khi người dùng đăng nhập, hash mật khẩu đã nhập, so sánh với mật khẩu đã lưu dưới database.

4. Hàm hash này **phải là hàm hash một chiều**, không thể dựa theo mật khẩu đã hash để suy ngược ra đầu vào.

```
1  public void Register(string username, string password)
2  {
3      string hashedPassword = HashHelper.Hash(password);
4      Database.SaveUser(username, hashedPassword);
5  }
6
7  public bool Login(string username, string password)
8  {
9      string pw = Database.GetHashedPasswordByUsername(username);
10     return pw == HashHelper.Hash(password);
11 }
```

Cách này đảm bảo **chỉ người dùng biết mật khẩu của họ**, dù là lập trình viên hay database admin, có nắm được cả code lõi database cũng không tài nào mò ra mật khẩu. Tuy nhiên, cách này có một vấn đề: Hai mật khẩu giống nhau khi hash sẽ có kết quả giống nhau. Hacker có thể mò ra mật khẩu bằng cách dùng *dictionary attack* – hash toàn bộ các mật khẩu có thể trong từ điển, rồi so sánh kết quả với mật khẩu đã hash dưới database.

Thế nhưng, vỏ quýt dày có móng tay nhọn. Đây là cách lưu trữ mật khẩu đúng mà hiện nay các framework đều áp dụng:

1. Khi tạo mật khẩu, tạo random một chuỗi kí tự gọi là salt.
2. Salt sẽ được cộng vào sau mật khẩu, toàn bộ chuỗi mật khẩu và salt sẽ bị băm (hash).
3. Lưu salt và giá trị đã băm xuống database (Một người dùng sẽ có 1 salt riêng).
4. Khi người dùng đăng nhập, lấy salt của người dùng, cộng nó với mật khẩu họ nhập vào, hash ra rồi so với giá trị trong database.

```
1  public void Register(string username, string password)
2  {
3      string salt = SaltHelper.getRandomSalt();
4      string hashedPassword = HashHelper.Hash(password + salt);
```

```
4     Database.SaveUser(username, hashedPassword, salt);  
5 }  
6  
7 public bool Login(string username, string password)  
8 {  
9     string salt = Database.getSaltByUsername(username);  
10    string pw = Database.getHashedPasswordFromUsername(username);  
11    return pw == hash(password + salt);  
12}  
13
```

Với cách này, khi người dùng quên mật khẩu, hệ thống không tài nào mò ra mật khẩu để gửi cho họ. Cách giải quyết duy nhất là **reset mật khẩu**, random ra một mật khẩu mới rồi gửi cho người dùng.



Ối giời phức tạp thế, cùng lầm thì lộ password trên trang của mình thôi mà Nói nhỏ một bí mật (mà chắc ai cũng biết) cho các bạn nghe nè: Hầu như người dùng chỉ sử dụng 1 username/mật khẩu duy nhất cho toàn bộ các tài khoản trên mạng. Nếu hacker tìm được mật khẩu từ trang của bạn, chúng sẽ thử với các account facebook, gmail, tài khoản ngân hàng, ... của người đó. **Mất 1 account là xem như mất sạch sành sanh.** Kinh khủng chua!. Không tin à, bạn thử ngẫm lại xem, bạn có dùng chung 1 email/mật khẩu cho Gmail, Facebook, Evernote, ... và nhiều trang khác không?

Nói đi nói lại một hồi, cũng đến cái “Lỗ hổng bảo mật khủng khiếp của Lotte Cinema”

Một ngày đẹp trời nọ, mình định dẫn gấu đi xem phim, ăn uống rồi *beep*. Định đặt vé online mà quên mất mật khẩu lottecinema.com, mình mò mẫm phần đăng nhập, tìm hoài mới thấy mục “Quên mật khẩu”. Nhập địa chỉ mail và chứng minh nhân dân, mình mau chóng nhận được một email gửi từ lottecinema, trong đó có cả username và mật khẩu của mình (Mail gì đã cùt lùn lại còn sai chính tả -_-) .



Thật là tiện quá đi mất, khỏi phải reset mật khẩu. *Khoan, có cái gì sai sai ở đây!!* Vậy là **bạn lotte lưu thẳng mật khẩu của mình thẳng dưới database** à. Lõi database bị thất thoát dữ liệu là toàn bộ các tài khoản khác của mình (Và các thành viên lotte cinema khác) cũng đi tong theo. *Thật là đáng sợ!* Lỗi này mình phát hiện năm ngoái, đến cách đây mấy ngày vẫn còn y nguyên. Thế mới biết bộ phận IT của lottecinema giỏi giang thế nào. Các bạn có tài khoản lotte cinema thì nhớ cẩn thận nghe.
Tác giả không phải dân chuyên về bảo mật, hệ thống mạng nên có gì sai sót mong các bỏ qua và góp ý hộ nhé.

Thực trạng học lập trình của một số thanh niên hiện nay

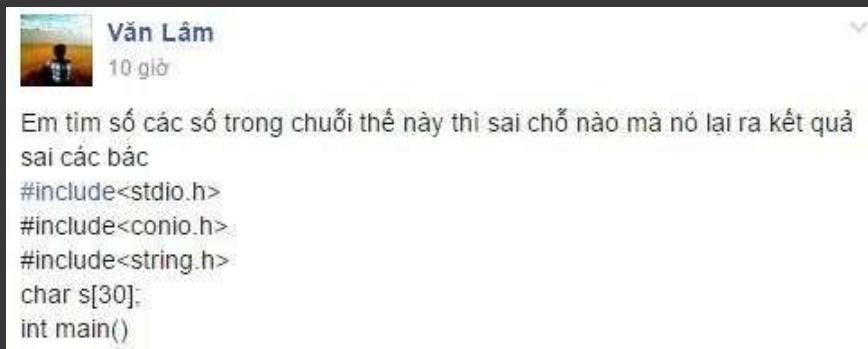
Posted on 17/12/2015 by Phạm Huy Hoàng

Lưu ý: Bài viết này được viết khi tác giả đang “hơi” xay xin và mất kiềm chế cảm xúc. Nội dung bài viết **đụng chạm chửi bới** đến khá nhiều đối tượng. Các bạn khi đọc thấy không dính dáng gì tới mình thì cứ đọc tiếp, còn nếu thấy hơi nhột thì cứ nghĩ là “chắc nó chưa mình ra” nhé.

Thực trạng học lập trình của các “sinh dzien”

Để quảng bá blog, mình tham gia khá nhiều group lập trình trên [facebook](#). Các bạn lập trình viên đang học hoặc mới ra trường cũng nên tham gia. Các group này thường đăng tin quảng cáo tuyển dụng, tìm lập trình viên, hoặc có các đường link tới các bài viết vô cùng bổ ích.

Tuy nhiên, điều khiến mình bức mình nhất là đa số các bạn lại sử dụng các group này để làm kenh... nhờ giải bài tập, fix bug, thi hộ (Bốc ảnh random chứ không có ý trù dập ai nhé =)).





Kupjd Le

6 giờ

Mọi người fix lỗi giúp mình với dk không a.Mình mới tập tò C# mà đang làm về phần mềm tiếng nhật có 2 form như sau.và nó báo lỗi như thế này.mong mọi người giúp đỡ

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
```



Thanh Long

23 Tháng 11 lúc 5:16

Chào mọi người khi public webiste trên iis có bị lỗi này ai biết fix giúp e với
The view 'Index' or its master was not found or no view engine supports
the searched locations. The following locations were searched:

```
~/Views/Home/Index.aspx
~/Views/Home/Index.ascx
~/Views/Shared/Index.aspx
~/Views/Shared/Index.ascx
```

Thường ngày thì mình cũng ngó lơ, xem như không có gì thôii, nhưng hôm nay úc ché
quá nên mới lên blog này để xả. Hồi khi làm việc còn đõ, mình thấy nhiều bạn
còn **post bài tập lên, nhò giải giùm**, hoặc **post đề thi lên kèm dòng chữ “Giúp em
với em đang thi”**. Thật lòng mà nói, mình cũng *éo biết các bạn trẻ này tốn tiền hoc
đai hoc làm gì nữa.

Học lập trình ở Việt Nam rất chán. Năm đầu bạn sẽ phải nhồi vào đầu mớ kiến thức
đại cương vô dụng, về sau mới được học lập trình. Số môn lập trình cũng rất ít, vì vậy
bài tập, bài thi là cơ hội để các bạn rèn luyện kĩ năng, luyện tư duy suy nghĩ. Hai trong
số các kĩ năng quan trọng nhất của lập trình viên là: kĩ năng fix bug và kĩ năng tu
hoc. Các bạn trẻ, **làm ơn bỏ cái thói** hẽ code không chạy, hẽ gấp bug là vác lên
group hỏi, rung đùi chờ người trả lời.

Group không phải là nơi fix bug chùa, thênh nhưng một số bạn lại rất thích làm “người
tốt việc tốt”, sẵn sàng bỏ thời gian viết code hộ, team viewer fix bug hộ. Các bạn
tưởng mình đang giúp người khác, nhưng thật ra làm vậy sẽ tạo thói quen ý lại, thu
chột kĩ năng của chính các bạn được giúp. Thời còn đi học, mình cũng biết nhiều bạn
cũng không chịu làm bài, tới lúc làm đồ án, gân ra trường thì lại **éo viết được 1 chức
năng nào**. Đây là hậu quả của việc nhò làm bài, nhò fix bug hộ.

Mà thôi kê, mấy đứa không code được ra trường thì không giành công ăn việc làm với mình được, càng đỡ phải lo. Lúc đi phỏng vấn chắc tụi nó cũng không kịp lên facebook hỏi hay nhờ người trả lời hộ đâu :3.



O cái thằng dở này, không hỏi ở đây thì hỏi ở đâu?

Như mình đã nói, kỹ năng tự học là một trong những kỹ năng quan trọng của lập trình viên. Trước khi hỏi thì các bạn làm ơn **chịu khó google trước** giùm mình, google tiếng Việt không ra thì google tiếng Anh nhé. Nếu tiếng Anh yếu hay không giỏi thì chịu khó học giùm mình cái, học reading thôi cũng được, ngành này mà tiếng Anh kém thì hơi khó sống.

Còn “bài tập hay fix bug thì google thế *éo nào được?”. O, bài tập với fix bug thì phải tự làm tự fix đi chứ, sau này ra trường đi làm bạn cũng nhờ người khác fix bug với code phụ à. Có nhiều trường hợp bất khả kháng, cần phải hỏi, thì các bạn vui lòng viết rõ ràng một tí. Có nhò kiểm tra code thì nhớ bỏ lên fiddle, dân tình lười lắm, không ai rảnh tải file về tìm lỗi cho bạn đâu.



Lời kết

Lấy [stackoverflow](#) ra làm ví dụ: Đây là trang hỏi đáp lớn nhất cho giới lập trình viên, các thành viên của họ đều rất ghét việc **nhờ giải bài tập, nhờ fix bug hộ**. Thay vì post câu hỏi lên group, hãy tự google trước đi, đa phần các lỗi/vấn đề bạn gặp đều có người trả lời trên trang này cả đấy.

Chửi một hồi thì cũng xả được cục tức rồi, bạn nào có muôn chửi chung thì cứ comment thoái mái cuối bài viết nhé. Hi vọng admin các group lập trình cũng đọc bài viết này, cùng chung tay xây dựng một cộng đồng lập trình viên: **Không thi hộ, không hỏi đáp bài tập, không hỏi nhảm** nhé.

Giới thiệu tổng quát về Meteor

Posted on 10/12/2015 by Phạm Huy Hoàng

Hỏi thật nhé, có bao giờ bạn thấy lập trình một ứng dụng Web là chuyện khó khăn? Chúng ta phải học cơ bản đủ thứ: Từ front-end như HTML, CSS, JS, cho tới back-end như Java/C#/Ruby, SQL, Nếu muốn làm thêm ứng dụng bản trên di động, ta phải học Objective C, Java, ... Sau khi code, ta còn phải tìm hiểu về hosting, về domain, về appstore để đưa web/ứng dụng của mình lên Internet, phiền phức nhiều khê quá nhỉ?

Có cách nào để học ít ngôn ngữ, nhưng lại mau chóng đưa ra thành phẩm không? Nếu chỉ là web đơn giản/e-commerce thì bạn có thể xây dựng với Joomla, Drupal, WordPress,... còn với những yêu cầu phức tạp thì **không có cách nào đâu.**



À, mà **thật ra là có đấy**. Chỉ cần bạn chịu khó học javascript, sau đó học Meteor là xong. Chỉ cần chút kiến thức về HTML, JS, CSS và MongoDB, bạn có thể xây dựng 1 ứng dụng web và mobile **realtime** trong 1 tiếng. (Nhanh gấp 5-10 lần Java, C#, PHP hay Rail để làm chuyện tương tự).

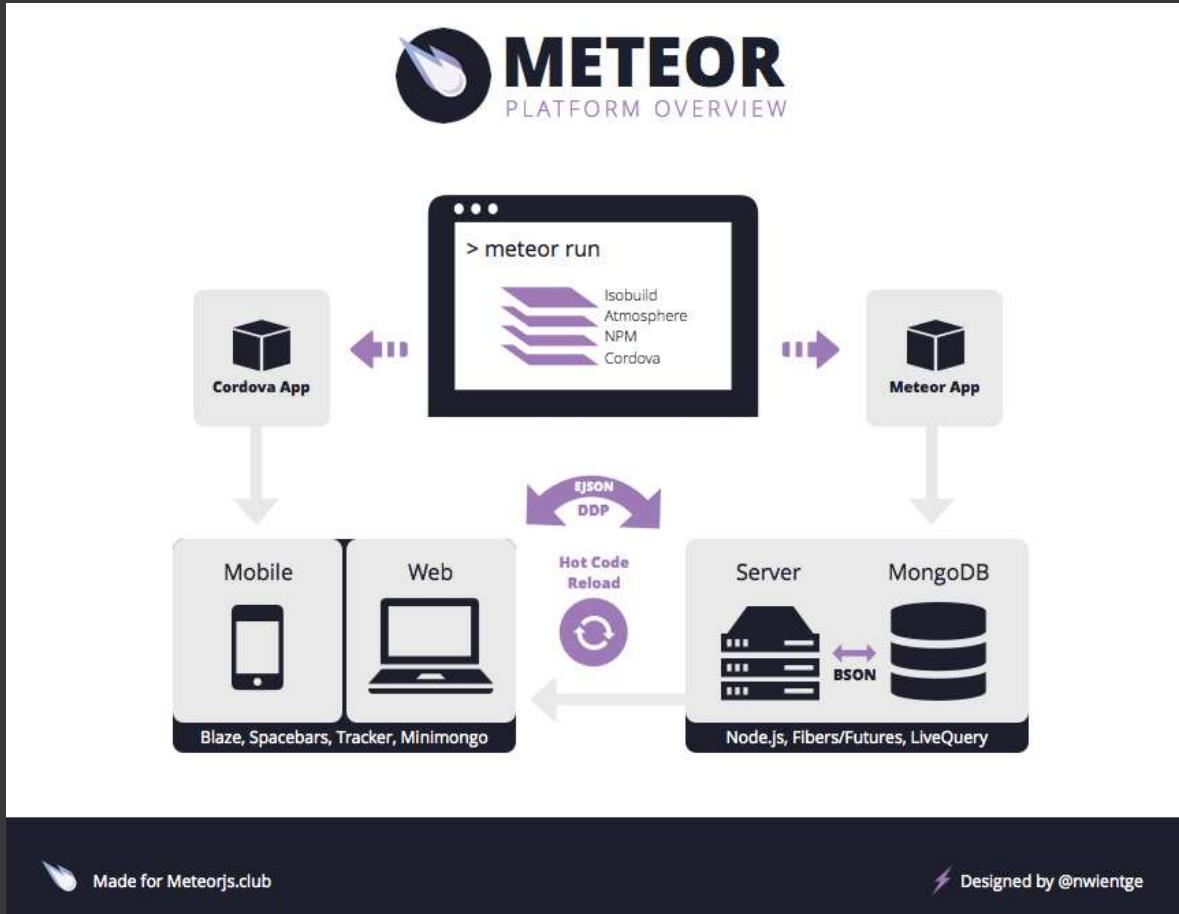
Không tin à? Hãy đọc bài viết để biết Meteor là gì, tại sao nó lại “bá đạo” như vậy nhé.

Meteor là gì?

Một số bạn hiểu là Meteor chỉ là một *thư viện/framework* như jQuery, AngularJS. Thật ra không phải vậy, Meteor không chỉ là 1 Javascript framework mà là **cả 1 hệ sinh thái** (ecosystem). Meteor cũng cung cấp cho ta “gần như” toàn bộ những thứ để làm một ứng dụng web/mobile:

- Phần front-end sử dụng framework Blaze để bind dữ liệu.

- Phần back-end sử dụng NodeJS và Express để làm server, MongoDB là database chính. Bạn không cần biết gì về NodeJS hay Express, chỉ cần code Meteor bằng javascript thôi.
- Một hệ thống thư viện/module tương tự npm, Maven, Nuget.
- Một số tool để build/deploy **web site** và **mobile app**. Chỉ cần code một lần, bạn có thể deploy nó thành *một web app*, hoặc build nó thành một *mobile app* trên Android, IOS.



Meteor có gì hot?

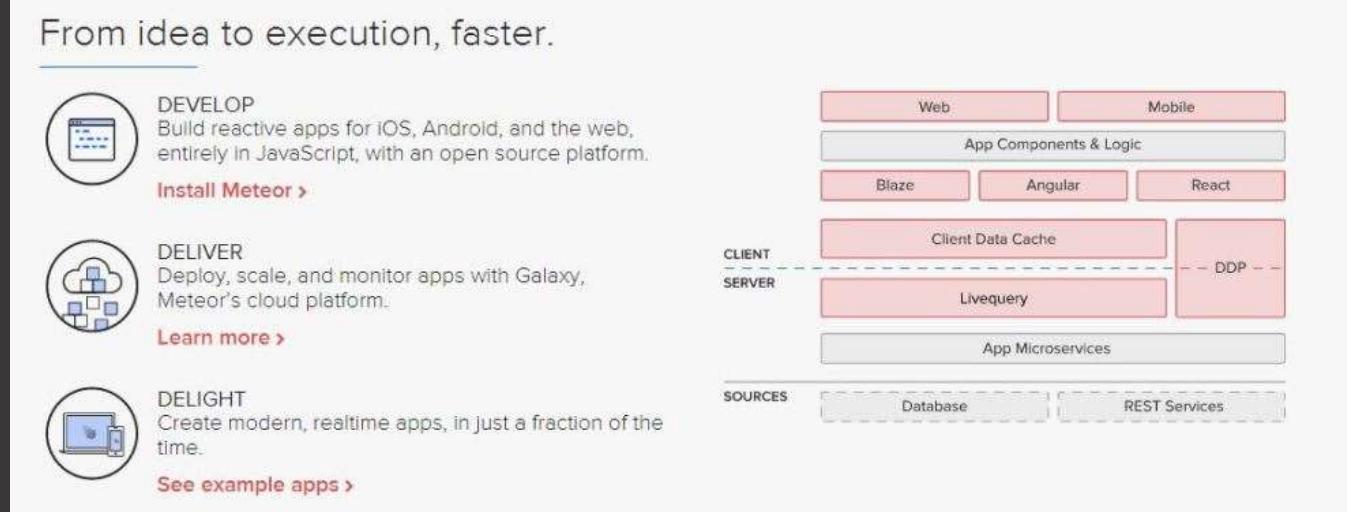
Vậy tại sao Meteor lại trở nên hot như vậy? Sử dụng Meteor, việc phát triển một ứng dụng sẽ trở nên **dễ dàng, nhanh chóng** hơn. Meteor được tạo ra không phải chỉ cho developer, mà cho những người “*bình dân*” như anh Ba bán cháo lòng, chị Bảy chủ tiệm nail, anh Sáu chủ quán cafe có thể **tự làm trang web/ứng dụng** cho doanh nghiệp của mình. Trên thực tế, có khá nhiều bài viết về chuyện sinh viên đại học, chủ tiệm nail, thợ xăm,... xây dựng ứng dụng bằng Meteor.

Các lý do Meteor được ưa chuộng:

- **Không cần quá nhiều kiến thức lập trình**, chỉ cần HTML, CSS, javascript, chút kiến thức về OOP và database là được. Mọi thao tác tới database đều thông qua các API của Meteor, viết bằng javascript, chả cần phải biết SQL là gì.
- Meteor **hướng tới các đối tượng không giới lập trình**. Do đó, Meteor rất dễ học dễ làm, tài liệu về Meteor cũng rất phong phú và dễ tìm.
- **Nhanh chóng tạo ra sản phẩm**, được các công ty start-up, các doanh nghiệp nhỏ ưu chuộng (Ngày xưa Ruby on Rail cũng nổi tiếng nhờ lý do này).
- Có **vô số module đã viết sẵn**, chỉ cần gắn vào và sử dụng. Ví dụ việc đăng nhập, phân quyền khá phức tạp trong C#, Java, ... trong Meteor chỉ cần gắn module vào, chỉnh sửa một chút là được.
- Dân Developer cũng thích Meteor, vì nó **tích hợp đủ thứ công nghệ**: Node.js, Express, [MongoDB](#), WebSocket, Phonegap, Trời ơi, lại còn **realtime** nữa.
- Code ít, được nhiều, Deploy rất nhanh. Vừa code xong và muốn đưa lên web, chỉ cần “*meteor deploy hoangph.meteor.com*”. Muốn có ứng dụng di động, chỉ cần “*meteor build android*” là xong.

Mình có làm thử một cái to-do-list realTime tại: <http://hoangphtodo.meteor.com/>.

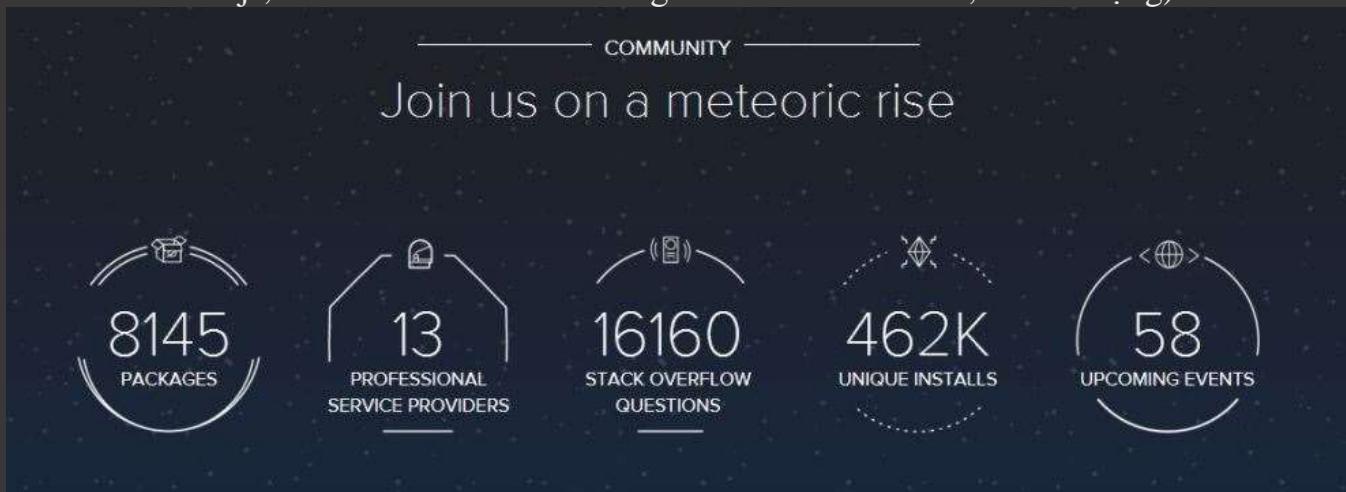
Việc sign-in chỉ cần khoảng 3 dòng code (Mới đăng nhập thông thường và google, facebook mình chưa đăng ký). Việc deploy cũng rất nhanh, việc phát triển toàn bộ ứng dụng chỉ mất 1 tiếng vừa học vừa làm.



Vẫn còn vài khuyết điểm

Đoạn đầu bài mình hơi *quảng cáo quá mức* cho Meteor. Thật ra, có một số mặt nó làm không tốt bằng các phương thức lập trình cũ với C#, Java, Ruby, ... Nó vẫn còn một số khuyết điểm dưới đây:

- Chỉ mới hỗ trợ MongoDB làm database engine.
- View Engine Blaze khá cùi bắp. May là hiện tại ta có thể thay thế bằng [AngularJS](#).
- SEO cũng chưa tốt, do Meteor tập trung vào phát triển ứng dụng real time, tương tác cao. Nếu cần làm web bán hàng, blog, trang tin tức, PHP & Joomla hay WordPress vẫn là lựa chọn hàng đầu nhé.
- Module nhiều nhưng document không rõ ràng (Tình trạng tương tự như npm của Nodejs, có rất nhiều module nhưng document rất mờ hờ, khó sử dụng).



Có nên thử sức với Meteor?

Thật ra mình vẫn chỉ giữ thái độ dè chừng, chỉ sử dụng Meteor để xây dựng các ứng dụng nhỏ, vừa học vừa chơi vì nó còn khá non trẻ. Các công ty cũng hơi dè chừng khi sử dụng Meteor. Tuy nhiên, theo mình tìm hiểu thì tương lai của Meteor khá tươi sáng, đội ngũ phát triển Meteor được tài trợ hơn **11 triệu \$**. Cộng đồng Meteor cũng rất đông đúc và ngày càng phát triển.

Nếu bạn là lập trình viên, muốn mở rộng kiến thức, hãy thử học Meteor xem. Biết đâu 1,2 năm nữa nó sẽ trở thành trào lưu thì sao, các công ty thi nhau tuyển Meteor thì sao? Nếu bạn mới học lập trình web, Meteor cũng là một lựa chọn không tồi. Chỉ cần chút kiến thức HTML/CSS/JS, bạn có thể build một website, một ứng dụng di động, đâu có gì khó phải không nào.



Một số tài liệu khá hay cho bạn nào muốn tìm hiểu thêm:

<http://javascriptissexy.com/learn-meteor-js-properly/#more-1498>

<https://www.meteor.com/tutorials/blaze/creating-an-app>

<https://meteorhacks.com/meteor-js-web-framework-for-everyone>

<http://joshowens.me/what-is-meteor-js/>

<http://www.manuel-schoebel.com/blog/meteorjs-the-perfect-match-for-lean-startups>

Một số kĩ thuật design cơ bản cho developer

Posted on 28/01/2016 by Phạm Huy Hoàng

Hắn là có vài bạn (sinh viên) đang muốn hỏi: Ông tui là lập trình viên, **lo code thôi chứ có phải thiết kế gì đâu mà phải học design?** Xin trả lời là: trừ khi bạn đi theo con đường lập trình nhúng, hoặc làm back-end developer 100%, thế nào bạn cũng sẽ phải đụng tới vài việc liên quan tới thiết kế.

Nếu may mắn, những công việc như thiết kế giao diện web, giao diện di động sẽ được thực hiện bởi designer, developer chúng mình chỉ cần file HTML hoặc PSD và code thôi. Tuy nhiên, ở Việt Nam, nhiều khi developer chúng mình cũng phải kiêm luôn nhiệm vụ này.

Thiếu kiến thức về design, đôi khi dev chúng mình cho ra nhiều giao diện khá là **khủng khiếp** (Thuở xưa ở Đại Học cũng có dạy môn HCI – Tương tác người dùng, nhưng cũng chẳng có tác dụng mấy). Do đó, bạn hãy tự trang bị cho mình những *kiến thức design cơ bản*, để tạo ra những design *coi được*. Những kiến thức này sẽ rất hữu dụng khi bạn muốn đi theo con đường front-end developer, hoặc full-stack developer.



Lẽ dĩ nhiên, dân developer chúng mình **không khéo tay, sáng tạo như bè lũ designer** được. Chúng ta thiết kế không phải hướng tới cái đẹp, mà hướng tới **sự hữu dụng**, đẹp và thuận mắt là thứ yếu. Mình cũng biết, các bạn thích làm việc với HTML, CSS, JS chứ không phải InDesign hay Photoshop, ta không thể học design theo cách của tụi designer được.

May mắn thay, mình tình cờ lbum được một số bí kíp khá hay về **Design for developer**, kiến thức rất thú vị, trực quan, phù hợp cho giới developer. Dĩ nhiên là

mình không ăn một mình mà mang lên đây chia sẻ cho mọi người, mong các bạn ủng hộ. Bài viết chỉ nói về một số vấn đề cơ bản mà quan trọng: Màu sắc, font chữ, và layout.

Màu sắc

Màu sắc và logo là 2 thứ đầu tiên đập vào mắt người dùng khi họ sử dụng trang web/ứng dụng. Với Facebook đó là màu xanh dịu dàng, với Yahoo là màu tím mộng mơ. Chọn màu sắc là một việc tưởng dễ nhưng khá khó. Nếu không biết cách chọn, các màu sẽ đánh nhau chan chát, nhìn rất ngứa mắt. Xin giới thiệu với các bạn một công cụ mà lũ designer hay xài – vòng tròn màu **thần thánh**: <https://color.adobe.com/create/color-wheel/>.



Để chọn một bộ màu phù hợp cho website (Khoảng 5-6 màu), ta cần làm theo các bước sau:

1. Chọn màu chủ đạo (Dựa theo logo hay gì đó).
2. Tìm màu đó trên vòng tròn màu. Nếu muốn sử dụng một tông màu duy nhất, dùng chế độ Analogous hoặc Monochromatic. Nếu muốn có các tông màu tương phản, dùng chế độ Triad, hoặc Complementary.
3. Nghịch ngợm vòng tròn màu cho tới khi tìm được bộ màu phù hợp. Nếu bí, các bạn có thể tham khảo một số bộ màu được đánh giá cao ở đây: <https://color.adobe.com/explore/most-popular/?time=all>.

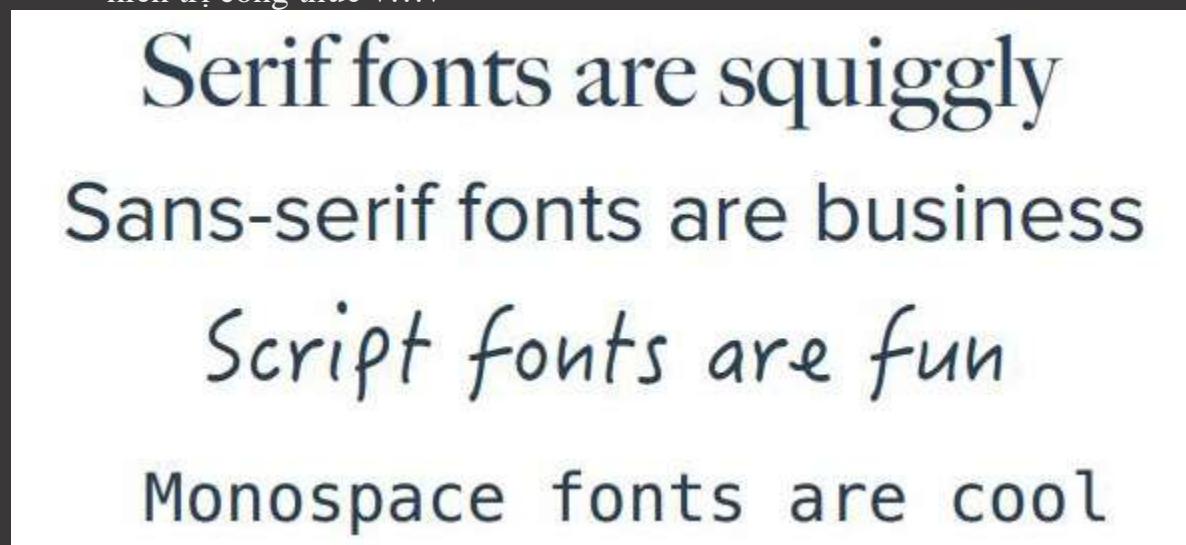
Dĩ nhiên, đằng sau chuyện chọn màu này là một đồng lý thuyết về thị giác, tâm lý,...
Mà kệ nó, lũ designer phải học mấy cái đấy, còn mình biết cách để lấy màu đẹp là
được rồi.

Font chữ (Typography)

Font chữ chính là thứ làm sự chuyên nghiệp của website. Nhiều bạn thường chỉ dùng
mỗi Arial và Times New Roman để làm font, do đó nhìn website có vẻ khô cứng,
thiếu tự nhiên. Website của mình dùng Open Sans tạo cảm giác thanh mảnh dễ
đọc: <http://conanak99.github.io/>.

Có hàng ngàn font chữ được sử dụng, tuy nhiên chúng có thể được chia thành vài loại
dưới đây:

- Serif font: Font có các dấu gạch ở chân, cuối chữ, tiêu biểu là Times New Roman. Font này thường được dùng khi in ấn, hoặc cho nội dung vì chúng khá trang trọng, dễ đọc.
- San-serif font: Font không chân, tiêu biểu là Calibri, Verdana. Font này thường được dùng để hiển thị giao diện, title, button, vì chúng dễ đọc dù ở size nhỏ.
- Script font: Font viết tay, thường dùng để trang trí, tạo cảm giác nhẹ nhàng vui nhộn. Nếu không biết dùng thì đừng dùng nhé.
- Monospace font: Các chữ trong font này có cùng bề ngang, thường dùng để hiển thị công thức v...v



Các bạn có thể tìm font đẹp tại Google Font, sau đó tìm hiểu cách kết hợp các font
tại <http://hellohappy.org/beautiful-web-type/> và <http://femmebot.github.io/google->

type/nhé. Điều quan trọng nhất, để người dùng dễ đọc, nhớ đừng để chữ nhỏ quá (Khoảng 16px là vừa), tăng cỡ chữ cho web di động nhé.

Layout

Các designer chuyên nghiệp thường sử dụng grid layout, layout theo dạng khối. Layout thường được chia thành 12 cột .12 là con số thần thánh vì nó chia hết cho 2, 3, 4, 6, designer có thể *chia màn hình thành 2, 3 hay 4 cột đều được*. CSS framework nổi tiếng là bootstrap cũng áp dụng con số 12 này vào grid system của họ:<http://getbootstrap.com/css/#grid>.

Một lời khuyên nữa từ các designer là **đừng tiết kiệm khoảng trắng**. Trong thiết kế, khoảng trắng tạo cảm giác thoáng đãng, rộng rãi, xa hoa. Thử ghé thăm Google hoặc Apple, bạn sẽ thấy giao diện rất trực quan, dễ nhìn với nhiều khoảng trắng. Tương phản là một số trang web bán hàng hoặc tin tức (Yahoo), nhồi nhét đủ thứ vào rất chật chöt, tù túng và rối mắt. Nhắc lại lần nữa, **đừng tiết kiệm khoảng trắng** (Nhưng đừng để website trông huo trông hoác nhá).



Bài học quan trọng nhất trong thiết kế là: **Khi không biết phải làm gì, hãy tham khảo và “chôm chia”**. Mục đích thiết kế của dân developer, **không phải để tạo ra cái đẹp, cái mới**, mà là **để tạo ra sản phẩm tốt nhất**, dễ sử dụng nhất. Do đó, cứ thoái mái học hỏi và chôm chia từ cái design đỉnh nhé.

Mọi người thảo luận và chia sẻ với nhau các tài liệu học design trong phần comment nhé. Bạn nào là designer hoặc chuyên về design có thể ném gạch, góp ý hộ để bài viết của mình được đúng đắn, chính xác hơn, mình xin nhận gạch đá nhiệt tình luôn.

Một số tài liệu được tham khảo trong bài viết:

- <https://speakerdeck.com/fcoury/design-for-developers>

- <https://speakerdeck.com/sugarenia/design-for-developers>
- <https://speakerdeck.com/smbarne/design-101-for-developers>
- <https://boagworld.com/design/why-whitespace-matters/>

Series JavaScript sida – Luận bàn về cái đít (this) trong javascript

Posted on 26/01/2016 by Phạm Huy Hoàng

Trước đây, mình đã đề cập tới một khái niệm khá khó nhằn trong javascript là callback. Ở bài viết này, mình sẽ giải thích về cái **đít**, nhầm, cái **this** – một từ khóa dễ làm đau đầu các lập trình viên js. Ánh trong bài viết chỉ có tính chất minh họa, các bạn **chăm chú đọc bài chứ đừng ngắm mông với đít nhé ;)**.



Từ khi đít (this) nhỏ còn hiền lành vô hại

Khi mới học, ta thấy *this* cũng khá đơn giản và vô hại. Nếu bạn từng học Java hoặc C#, hẳn bạn cũng nhớ **từ khóa this dùng để trỏ tới chính object gọi hàm đó**. Trong javascript, từ khóa this cũng đóng vai trò tương tự. Ở dòng code dưới, ta sẽ thấy *this* trỏ tới object *person*, in ra đúng những điều ta muốn.

```
var person = {  
    firstName: 'Hoang',  
    lastName: 'Pham',  
    showName: function() {  
        console.log(this.firstName + ' ' + this.lastName);  
    }  
}
```

```
});  
//Ở đây this sẽ là object person  
person.showName(); //Hoang Pham.
```

[view rawthisSample.js](#) hosted with [GitHub](#)

Một trường hợp khác, khi ta khai báo biến global và hàm global, toàn bộ các biến và hàm đó sẽ nằm trong một object có tên là *window*. Lúc này, khi ta gọi hàm *showName*, chính object *window* là object gọi hàm đó, *this* trỏ tới chính object *window*.

```
var firstName = 'Hoang', lastName = 'Pham';  
// 2 biến này nằm trong object window  
  
function showName()  
{  
    console.log(this.firstName + ' ' + this.lastName);  
}  
  
window.showName(); // Hoang Pham. this trỏ tới object window  
showName(); //Hoang Pham. Object gọi hàm showName vẫn là object window
```

[view rawthisGlobalSample.js](#) hosted with [GitHub](#)

Tới khi đít lớn và gây bao rắc rối

Nếu chỉ sử dụng theo 2 cách nêu trên, *this* khá dễ hiểu vào không gây ra mấy khó khăn khi sử dụng. Song, sự đáng sợ và khó chịu của *this* sẽ dần lộ ra qua các ví dụ dưới đây. Các bạn vui lòng tiếp tục **chăm chú đọc bài chứ đừng ngắt mông với đít nhé ;)**.



Rắc rối 1 – Hàm được truyền vào như một callback

Giả sử, ta muốn khi người dùng click vào một button, ta sẽ gọi hàm `showName` của user. Vô cùng đơn giản, ta chỉ cần truyền hàm `showName` vào như một callback cho hàm `click` là xong.

```
var person = {  
    firstName: 'Hoang',  
    lastName: 'Pham',  
    showName: function () {  
        console.log(this.firstName + ' ' + this.lastName);  
    }  
};  
  
//Ở đây this sẽ là object person  
person.showName(); //Hoang Pham.  
  
$( 'button' ).click( person.showName ); //showName truyền vào như callback
```

Tuy nhiên, hàm lại không chạy như ta mong muốn. Mở [developer tools](#) lên thì thấy object *this* không có trường *firstName* và *lastName*. Kiểm tra kĩ chút nữa thì ta thấy *this* ở đây là chính button ta đã click vào, chứ không còn là object person như ví dụ trên nữa.

Trong trường hợp này, ta có thể sửa lỗi bằng cách sử dụng anonymous function, hoặc dùng hàm *bind* để xác định tham số *this* cho hàm truyền vào là được.

```
var person = {  
    firstName: 'Hoang',  
    lastName: 'Pham',  
    showName: function () {  
        console.log(this.firstName + ' ' + this.lastName);  
    }  
};  
  
$('button').click(person.showName); //showName truyền vào như callback, ở đây this chính là button  
  
// Dùng anonymous function  
$('button').click(function () { person.showName(); });  
  
// Dùng bind  
$('button').click(person.showName.bind(person)); //this ở đây vẫn là object person
```

[view rawthisSampleCallbackFixed.js](#) hosted with [GitHub](#)

Rắc rối 2 – Sử dụng *this* trong anonymous function

Giả sử, object person có một danh sách bạn bè, bạn muốn viết một function show toàn bộ bạn bè của person đó. Theo lý thuyết, ta sẽ viết như sau:

```
var person = {  
    firstName: 'Hoang',  
    lastName: 'Pham',  
    friends: ['Thien', 'Hieu', 'Duc'],  
    showFriends: function () {  
        var str = '';  
        for (var i = 0; i < this.friends.length; i++) {  
            str += this.friends[i] + '  
        }  
        return str;  
    }  
};  
  
$('button').click(person.showFriends);
```

```

lastName: 'Pham',
friends : ['Minh', 'Sang', 'Khoa', 'Hoang'],
showFriend: function() {
    for var i = 0; i < this.friends.length; i++)
        console log this.firstName + ' have a friend named ' + this.friends[i]);
},
showFriendThis: function() {
    this.friends.forEach(function(fr){
        console log this.firstName + ' have a friend named ' + fr);
    });
};

person.showFriend(); //Hàm chạy đúng

person.showFriendThis(); // Hàm chạy sai

```

[view rawthisAnonymousSample.js](#) hosted with [GitHub](#)

Với hàm *showFriend*, khi ta dùng hàm for thường, hàm chạy đúng như mong muốn. Tuy nhiên, trong trường hợp dưới, khi ta dùng hàm *forEach* (xem lại ở [đây](#)), truyền vào một anonymous function, *this* ở đây lại thành object *window*, do đó trường *firstName* bị undefined.

Trong trường hợp này, cách giải quyết ta thường dùng là tạo một biến để gán giá trị *this* vào, và truy xuất tới giá trị đó trong anonymous function.

```

var person = {

    firstName: 'Hoang',
    lastName: 'Pham',
    friends : ['Minh', 'Sang', 'Khoa', 'Hoang'],
    showFriendFixed: function() {
        var personObj = this //Gán giá trị this vào biến personObj
    }
};

```

```
this.friends.forEach(function(fr){  
    console.log(personObj.firstName + ' have a friend named ' + fr);  
});  
}  
};  
  
person.showFriendFixed(); //Hàm chạy đúng
```

[view rawthisAnonymousSampleFixed.js](#) hosted with [GitHub](#)

Rắc rối 3 – Khi hàm được gán vào một biến

Trường hợp này ít xảy ra, nhưng mình cũng liệt kê cho đủ bộ. Đó là trường hợp khi ta gán một hàm vào một biến, sau đó gọi hàm đó. Hàm sẽ không chạy như ta mong muốn, vì object gọi hàm lúc này chính là object *window*.

```
var person = {  
  
    firstName: 'Hoang',  
    lastName: 'Pham',  
    showName: function() {  
        console.log(this.firstName + ' ' + this.lastName);  
    }  
};  
  
//Ở đây this sẽ là object person, chạy đúng  
  
person.showName(); //Hoang Pham.  
  
  
var showNameFunc = person.showName; //Gán function vào biến showNameFunc  
  
showNameFunc(); //Chạy sai, ở đây this sẽ là object window
```

[view rawthisVariable.js](#) hosted with [GitHub](#)

Để giải quyết, ta cũng sử dụng hàm *bind* như trường hợp trên cùng, quá đơn giản phải không nào?

```
var person = {  
    firstName: 'Hoang',  
    lastName: 'Pham',  
    showName: function() {  
        console.log(this.firstName + ' ' + this.lastName);  
    }  
};  
  
var showNameFunc = person.showName.bind(person); //Gán function vào biến showNameFunc  
  
showNameFunc(); //Chạy đúng vì this bây giờ là object person, do ta đã bind
```

[view rawthisVariableFixed.js](#) hosted with [GitHub](#)

Trong bài viết, mình đã sử dụng hàm *bind* để giải quyết một số vấn đề với từ khóa *this*. Thật ra, liên quan tới cái đít (*this*) **không chỉ có bind mà còn có 2 mĩ nữ khác cũng phức tạp không kém** là *apply* và *call*. Ở một số bài viết tiếp theo, mình sẽ cùng các bạn lột đồ, lộn, vạch trần sự phức tạp của 3 hàm này. Nhớ đón xem nhé, ở các bài viết sau không có mông cũng chả có đít nữa đâu.



Bài viết có tham khảo từ nguồn: <http://javascriptissexy.com/understand-javascripts-this-with-clarity-and-master-it/>. Đây là một trang web khá hay về javascript, mình ôn lại được khá nhiều kiến thức js bị hỏng trong này.

Sự thật đáng lòng: Đôi khi cắm đầu ngồi CODE là cách ... ngu nhất để giải quyết vấn đề

Posted on 20/01/2016 by Phạm Huy Hoàng

Qua bài viết về [button trị giá 300 triệu đô](#), mình thấy các bạn có vẻ hào hứng với những bài viết theo phong cách kể chuyện. Vì vậy, ở bài viết này, mình sẽ bắt đầu bằng cách kể một câu chuyện nho nhỏ về 1 chàng **coder** nghèo tên K (Gọi là Khoa Khoé Khoang hay Khải gì đó tuỳ bạn).

Tiếp xúc với máy tính từ năm 10 tuổi, K vô cùng ngạc nhiên trước sức mạnh của cỗ máy vô tri vô giác ấy, và nuôi mơ ước trở thành một lập trình viên. Lên cấp 3, nhờ giỏi Toán, K được vào lớp chuyên Toán của trường. Với niềm đam mê lập trình, K nhanh chóng tiếp cận và thành thạo Pascal, C, giật được vài giải Olympic tin học.

Nhờ điểm cao, K đậu vào một [trường đại học](#) công khá danh tiếng. Vào trường, được học thêm Ngôn ngữ lập trình, về Cấu trúc dữ liệu và thuật toán, K càng ngày càng thích code hơn. K code ngày code đêm, cắm đầu vào luyện thuật toán cho thành guru, lúc rảnh rỗi K lại kiểm sách bài tập làm... cho đỡ thèm. K luôn nộp bài sớm hơn các bạn để thể hiện sự hơn người của mình. Do suốt ngày chỉ biết cắm mặt vào máy tính, K trải qua 4 năm đại học mà vẫn FA...



Tốt nghiệp ra trường, những tưởng một người **lập trình tốt, giỏi technical** (Như K tự nhận xét mình) sẽ kiếm được việc ngon, lương cao, [dễ dàng thăng tiến](#). Vòng phỏng vấn diễn ra cũng khá suôn sẻ, K trả lời trọn vẹn một số câu hỏi về technical, nhưng lại

ngập ngừng ở 1 số câu hỏi kiểu “Khi có xung đột với thành viên trong nhóm, em xử lý thế nào?” hoặc “Em đã có sản phẩm cá nhân nào nổi bật chưa?”.

Đồng nghiệp của K có một thằng tên là H. K rất khinh thằng H này vì nó chỉ tốt nghiệp ĐH dân lập (FPT), đã thế trong giờ làm còn ít ngồi code mà toàn vào [stackoverflow](#), pluralsight, webtretho, lâu lâu còn mở wordpress gõ gõ gì đó, hình như là [blog](#). Thế rồi, vì vài chuyện nhỏ nhặt dưới đây, K dần dần chuyển từ khinh sang ghét và căm thù thằng H :

- **Chuyện thứ nhất:** Công ty có một chương trình khá cũ chạy trên một con server cùi bắp. Gần đây, vì lưu lượng sử dụng nhiều, lâu lâu chương trình lại bị OutOfMemoryException (tràn bộ nhớ). Tự tin với khả năng code của mình, K xin anh team leader source code của chương trình để ngồi optimize lại. Hỉ hục mất 2 tuần, K chỉ mới optimize được 30% chương trình cho tiết kiệm RAM. Thằng H nghe chuyện, nhanh nhau đoán liên hệ với anh team leader và bên IT, lắp thêm 2 thanh RAM 4GB vào con server cũ. **Chỉ mất 5 phút**, từ đấy chương trình **chạy vo vo không còn OutOfMemory** nữa.



- **Chuyện thứ hai:** Team cần làm một phần mềm về phim ảnh, [trích xuất dữ liệu](#) từ Galaxy Cinema, Lotte, CGV. Với tinh thần hăng hái, ham học hỏi, K tự nghiên cứu về [web parser](#) để trích xuất thông tin từ các trang này. Do các trang này có dung Ajax, Paging phức tạp, K mất gần 2 tuần để viết và test việc parse dữ liệu từ Galaxy và Lotte. Công sức của K tiếp tục đổ sông đổ biển khi ngay hôm sau, thằng H nhanh nhau nói với anh team leader: “Anh ơi, bên 123phim nó có API sẵn rồi, có cả Cinebox với 4,5 hàng nữa cơ. Phí chỉ có 50k/tháng

thôi, mình liên hệ mua rồi integrate vào nha anh” và anh team leader gật đầu đồng ý.

- **Chuyện thứ ba:** K thương thầm một em QC da trắng dáng cao tên ngực khủng tên M cùng công ty. Ngoài làm QC, em còn chuyên *đăng hình khoe thân kiêm bán kem trộn làm trắng da* trên facebook. Nghe tiếng K code giỏi, em thủ thi “Anh K làm cho em cái trang web bán hàng nha, làm đẹp đep em **thưởng** cho <3”. Vốn dại gái, lại tự tin rằng mình giỏi Java thần thánh, K bắt đầu vẽ database, dùng Spring Boot, Heroku để code một trang bán hàng **hoành tráng**.
- Hôm sau lên công ty, đang định show giao diện cho em M xem thì K thấy em đang đứng bên cạnh thằng H, mắt long lanh đầy ngưỡng mộ “Anh H giỏi quá, làm web **vừa đẹp lại vừa nhanh**, em nhờ hôm qua mà giờ đã xong rồi”. Hóa ra thằng ku dùng PHP – thứ ngôn ngữ rẻ tiền, dung framework Magento và chôm đại cái theme có sẵn đâu đó để làm web cho em một cách nhanh chóng. Hết giờ làm, bé M **thưởng** thằng H bằng cách đi ăn uống, xem phim rùi đi chõ-mà-ai-cũng-biết-là-chõ-nào-đấy :>. Sau hôm đó, K nhìn thằng H bằng ánh mắt mang hình viên đạn.



Bé M (Ảnh minh họa)

- **Chuyện cuối cùng:** Cay cú với thằng H, K quyết định viết một [blog cá nhân](#) để cạnh tranh với nó. Thấy thằng ku dùng wordpress cùi bắp, K quyết tự khẳng định bản thân bằng cách code luôn một blog và publish mã nguồn lên [github](#) cho nó biết mặt. Code gần 3 tháng mà **vẫn chưa xong**, blog thì chưa viết được dòng nào, còn blog kia của thằng H đã được hơn [200 nghìn lượt xem](#).

Câu chuyện trên là hư cấu và không dựa trên bất cứ phần trăm sự thật nào (Bác nào thấy nhọt thì cứ việc). Hãy khoan chửi em M là con bitch ngực to, hay ông team leader ngu chết mẹ không thấy được khả năng của K. Rõ ràng, cùng làm một việc, nhưng H lại hoàn thành một cách nhanh chóng và nhẹ nhàng hơn K. Lý do là vì **K tiếp cận vấn đề theo cách của một coder, H lại tiếp cận vấn đề theo cách của một developer.**



Với coder, **code là thứ tối thượng có thể giải quyết mọi vấn đề**. Họ thích [tự học những ngôn ngữ/công nghệ mới](#). Họ là những người tập trung công sức để viết những dòng code thật tốt, chạy thật nhanh, optimize thật kĩ. Họ không thích tái sử dụng code của người khác, mà thích tự code để “học hỏi và phát triển”. Thế nhưng, một trong những sự thật đau lòng của ngành lập trình là đôi khi [éo ai quan tâm đến code mình viết cả](#).

Khác với coder, *developer thường tiếp cận vấn đề theo cách khác*. Họ cũng có thể code và code giỏi, nhưng họ tập trung trước nhất **vào vấn đề và cách giải quyết**. Khi

có yêu cầu từ khách hàng, họ sẽ phân tích, tìm cách giải quyết phù hợp nhất, ít tốn thời gian và công sức nhất, trước khi bắt tay vào code.

What is the difference between a coder (“code monkey”) and a software developer?



Như slogan “**từ coder đến developer**” của blog, mình viết để chia sẻ những kiến thức về technical và một vài thứ khác để giúp bạn **tiến hóa từ coder lên developer**. Giới technical sẽ giúp bạn làm một coder giỏi, nhưng chỉ technical không thôi là chưa đủ. Bạn cần phải nhét vào đầu tư duy lập trình, tư duy sản phẩm, tư duy phân tích thiết kế và vô số thứ khác để trở thành một developer đúng nghĩa. Để học được những điều này, hãy like fanpage và theo dõi những bài viết tiếp theo của blog nhé. Thân chào.

Chuyện bên lề, có 1 bạn khá dễ thương phỏng tác câu chuyện của mình theo 1 cách nhìn khác, cũng rất đáng đọc. Chắc cũng đồng cảm với bạn K nên trong truyện này bạn cho K 1 cái happy ending rất đẹp :D

Câu chuyện hôm nay kể về một chàng coder H. Cậu đến với tin học hoi muộn, và khi nhận ra mình đam mê tin học thì đã muộn cho việc chuẩn bị thi vào các trường đại học danh tiếng về CNTT. H không giàu, nhưng cũng đủ khéo để theo đuổi đam mê với đại học FPT.

Vào trường, cậu nhận thấy các trường đại học có vẻ danh tiếng kia dạy thật thiếu bài bản, toàn dạy những thứ lý thuyết, rất thiếu kỹ năng mềm, hay những thứ ứng dụng. Cậu học vừa đủ những môn CTDL, và tập trung hơn vào những môn “thực tế”, tự học lập trình ứng dụng và tham gia các hoạt động ngoại khóa.

Tốt nghiệp ra trường, nhò vón kỹ năng mềm cùng khả năng lập trình ứng dụng, cậu được nhận vào công ty. Nhưng khi phỏng vấn, cậu lại bị ngắt ngứa những câu cơ bản về cấu trúc dữ liệu, về nguyên lý lập trình.

Đồng nghiệp của H có một thằng tên K. H rất ghét thằng này vì nó tốt nghiệp đại học “danh tiếng”, lại còn khoe mấy cái giải thưởng tin học **toàn mớ lý thuyết vô nghĩa**. Hai đứa ngồi gần nhau, cậu lâu lâu ngó sang nó thấy nó cứ cầm cuộn viết code, chẳng quan tâm gì đến thế sự cả. Còn cậu vừa phân bố thời gian để hoàn thành deadline, vẫn có thời gian để lén webtretho, viết blog, lâu lâu **gặp bug thì chỉ cần lén stackoverflow là xong**. Cậu không thích thằng K và sẽ cố gắng chứng tỏ mình hơn K. Và cậu đã làm được như vậy.

Chuyện thứ nhất – thứ hai – thứ ba – tương tự bài gốc

H rất hài lòng với những gì mình làm được. Còn K vẫn cứ ngồi đó cố gắng viết code hoàn chỉnh cho cái website bán hàng và blog, thật là rỗi việc.

- Đến một ngày, cái phần mềm cũ rich chạy trên server cũi bắp lại dở chứng. Trưởng nhóm nhận thấy vấn đề diễn ra ngày càng thường xuyên hơn trước, tỉ lệ thuận với lượng dữ liệu đổ vào chương trình càng nhiều theo thời gian, và module gây ra chuyện đó còn được sử dụng ở nhiều chương trình khác nữa. Chẳng lẽ lại phải mua hẳn server mới và một loạt server khác nữa phòng tránh chuyện này? K nhanh nhảu chạy vào và xin tiếp tục dự án **“tối ưu hóa code”** đã bị dừng lại trước đó. Trưởng nhóm đồng ý, và dù rất khó khăn nhưng 2 tuần sau K cũng đã hoàn thành. Và tất cả các chương trình sử dụng chung module đó đều chạy nhanh và ổn định hơn trước. K được anh trưởng nhóm hết sức khen ngợi và hứa hẹn sẽ tăng lương vào đợt tiếp theo.
- 123phim **đột ngột dừng hoạt động** do kinh doanh thua lỗ. Các api chỉ còn truy cập được trong vòng 1 tháng. May mắn thay, nhò có **“crawler”** trước đây của K, nhóm đã nhanh chóng thay đổi và tích hợp trở lại vào phần mềm.
- Em QC lăm chuyện đòi H thêm chức năng cho bán hàng, lần này là **tích hợp với giaohangnhanh** để em đỡ mất công tìm người giao hàng, nhập đơn hàng bằng tay. Chết thật, H lâu lăm có dụng đến Magento đâu cơ chứ, lại chả có module nào trên mạng, mà cái theme cũng là hàng mua biêt customize sao bây giờ. QC giận H và chạy sang nhò K. K (dù rất ghét con nhò này) vui vẻ nhận

*lời vì K **năm** rất rõ code của mình và việc chỉnh sửa không quá tốn thời gian lắm. Từ đó con nhỏ QC này suốt ngày bám lấy K mà quên H.*

H tức giận vì K càng ngày càng được nhóm đề cao và cả em QC cũng đỗ theo hắn nữa. Giờ H chỉ còn một blog. Chắc, 200k lượt xem, tên K kia đến bao giờ mới đạt được. Hắn chỉ có lèo tèo vài bài viết chán ngòm về kỹ thuật. Đến một hôm, K xin phép nghỉ việc. H mừng lắm, cuối cùng cái gai trong mắt cũng chịu đi.

*Lại nói về K. Cái blog mà K chăm chút, viết blog dù út nhưng chất lượng may mắn được một nhà tuyển dụng để mắt tới. Và công ty của nhà tuyển dụng này thuộc hàng top thế giới. K nhận được thư mời, dù rất đắn đo và lo lắng nhưng cậu quyết định thử sức. Bất ngờ thay, công ty nợ chỉ toàn hỏi những câu hỏi liên quan đến giải thuật, tuy mà không hỏi bao nhiêu về công nghệ cả, và họ nói rằng họ rất ấn tượng với những giải thưởng tin học K đạt được. Tự tin với **kiến thức nền tảng** của mình, K trả lời xuất sắc hầu hết câu hỏi của nhà tuyển dụng. K nhanh chóng được đề nghị vào làm với một mức lương và môi trường làm việc mà mọi dev đều mơ ước.*

Developer cũng nên học ... marketing – Chiến thuật bán hàng thú vị của John Sonmez

Posted on 14/01/2016 by Phạm Huy Hoàng

Bạn nào theo dõi blog từ những ngày đầu sẽ biết mình khá thầm tượng anh chàng John Sonmez, tác giả blog [simpleprogrammer](#). Những bài viết trên blog của anh đã cho mình nhiều bài học vô cùng quý giá, truyền cảm hứng cho mình viết nên blog [Tôi đi code đạo](#) này. Blog của mình cũng đang đi theo con đường của anh này, truyền đạt những kiến thức về [lập trình](#), về [nghề nghiệp](#), giúp các bạn xác định [con đường cho bản thân](#).

John Sonmez cũng xuất thân là “developer quèn” như chúng ta. Tuy nhiên, hiện nay anh không còn đi “code đạo” như mình, mà chỉ làm consulting, có thu nhập bị động (Hơn 500.000/\$ năm nhờ các khóa giảng dạy trên [pluralsight](#), bán sách và một số khóa học trên website). Có thể nói anh là một trong những lập trình viên khá thành công. John cho rằng ngoài kỹ năng lập trình, kỹ năng **marketing** là một trong những thứ quan trọng nhất mà mọi lập trình viên nên có; biết cách marketing bản thân sẽ làm bạn [cao giá](#) hơn, dễ tìm được công việc lương cao như ý muốn.

I totally identify with Mike, because a few years ago, that was me.

My name is John Sonmez. I've been writing code for more than 25 years.

I'm also a software consultant to companies like Hewlett Packard and Verizon (who pay upwards of \$500 per hour for access to me)...

A career coach to thousands of developers through my Simple Programmer website (which receives more than 1.8 million visits a year)...

And the guy who's produced more online training for software developers than anyone else alive.



John Sonmez has published more online developer training than any other programmer.

Vừa tuần trước đây, mình bị dính **một cú marketing khá bất ngờ và thú vị** từ anh chàng này. Trò marketing này khá hiệu quả, mình thấy nó còn hay gấp mấy lần mấy cái quảng cáo, giới thiệu trên TV. Chưa biết anh code giỏi chừng nào nhưng khả năng bán thân, lộn, bán mình thật là bá đạo :D. Mình chia sẻ câu chuyện này với các bạn, nếu sau này **Bạn nào muốn bán hàng, bán thân hay bán dịch vụ** có thể áp dụng thử.

Từ một email úp úp mở mở

Vì nằm trong danh sách subscriber của blog Simpleprogrammer, lâu lâu mình vẫn nhận được mail từ blog, giới thiệu các bài viết hay (Các bạn cũng nên like trang [fanpage Tôi đi code dạo](#), hoặc đăng kí mail bên phải nhé). Vừa hôm trước, mình nhận được một email của John. Mọi chuyện bắt đầu từ đây.

Mail khá vui, kể về những ngày đầu anh vừa đi làm, đang làm tester với mức lương 25\$, anh apply qua một cty mới với mức lương 75\$. Phỏng vấn khá đơn giản: “Anh biết C++ không”, “À, biết”, thế là được nhận. Điều quan trọng ở đây là “Trước giờ anh chưa bao giờ đụng tới C++ cá”. John đã làm gì? Anh mua một cuốn [sách về C++](#), đọc lúc ăn lúc ngủ, đọc từ đầu tới cuối. Và rồi anh đã master C++ trong 1 tháng. ĐÙA ĐÁY, anh cũng chỉ là người thường như chúng ta thôi, không phải thiên tài đâu. Đọc hết cuốn sách, anh vẫn ngáo ngơ, chẳng thấm được mấy vào đầu. Và rồi John Sonmez rút ra được một chân lý:

Kỹ năng quan trọng nhất để sống sót cho developer là KỸ NĂNG HỌC TẬP.

From: John Sonmez (Simple Programmer) ★
Subject: You're fired!
To: Me <huyhoang8a5@gmail.com> ★

The real problem is that I was missing a critical skill. And even though this skill had nothing to do with C++ or any other programming language, I now consider this the #1 survival skill for ALL software developers.

That skill is:

The ability to LEARN EFFICIENTLY.

Now as developers, we tend to see ourselves as good at learning and adapting. I know I did.

But I was wrong. I wasted weeks of my time—and got fired for my trouble.

In fact, I didn't make much progress with my ability to learn for another 12 years. I wasted thousands of hours studying and retaining very little. I caused myself a lot of unnecessary stress and frustration, and missed so many opportunities...

Anh nói rằng sẽ chia sẻ, truyền đạt một số kinh nghiệm về học tập mà mình có được trong quá trình làm việc ở mail sau. Mình khá là háo hức về những điều anh sẽ nhắc đến ở email sau.

Thế rồi email thứ hai vào ngày hôm sau

Ở email này, John lại kể về một trải nghiệm khá thú vị. Anh bắt đầu thực hiện các khóa dạy trên pluralsight: C#, Java, SQL, Android, ... Thế rồi anh gặp một vấn đề: Mình hết cái để dạy cmnr. Cái khó ló cái khôn, anh quyết định: **Hết cái để dạy thì học ngôn ngữ mới rồi dạy thôi**. Anh quyết định học ngôn ngữ mới, và rồi ... thất bại đau đớn, vì việc học ngôn ngữ mới không dễ như anh từng nghĩ.

Không bỏ cuộc, anh vẫn ép buộc bản thân học và dạy. Vừa học, anh vừa tự rút ra những mánh khoe, những kĩ năng học tập. Đến khi kết thúc, anh có thể **học một ngôn ngữ mới**, quay một khóa học 4 tiếng chỉ trong vòng ... 10 ngày. (Học ở đây không phải là xem, đọc, biết sơ sơ, mà là có thể làm được, và **dạy được người khác** luôn nhé.). Một lần nữa, John lại úp mở về những kĩ năng này, anh hứa mọi thứ sẽ xuất hiện trong mail sau.

Subject: Learn or die

To: Me <huyhoang8a5@gmail.com> 

When you have this skill, you'll find that you enjoy your work more.

You have less stress.

And you have more confidence because your career is future-proof.

When I realized all this, I decided that I had to share what I'd learned with you—all of the shortcuts I'd discovered, all of the techniques that helped me master (and retain) new skills in such a short time.

I'm not ready to say more just yet. I'm still putting the finishing touches on everything with my team here...

Won't be long now—so keep an eye on your inbox!

Talk soon.

Email cuối cùng – Em đã bị lừa, bị dính đòn quảng cáo

Và rồi, ngay hôm sau, mình nhận được một email cuối cùng. John vừa tung ra khóa học “10 bước để học mọi thứ”. Một đường link trong email dẫn tới trang web quảng cáo cho khóa học này.

The Most Important Skill Any Software Developer Can Have Is... Knowing How to **LEARN**

And I'm not talking about just squirreling away knowledge for "someday."

Today's successful developer is a master at diving deep into a new piece of tech, slicing it into bite-sized chunks and absorbing the critical 20% that lets him work productively while other developers are still scratching their heads and searching Stack Overflow.

I have this ability today—but I didn't always.

A few years ago, learning any new technology felt like embarking on an endless trek to Mordor.

Here's what I'd do:

- ❑ Hit up Amazon and buy every book that looked remotely related.
- ❑ Pick up the first book I bought and plow right through from page 1 through to page 876.
- ❑ Repeat for 5-10 more books.

Trang web là một bài quảng cáo khá dài (nhưng rất hay) về khóa học. Bài quảng cáo còn phân tích những lợi ích bạn sẽ thu được so với số tiền bỏ ra khi mua khóa học, kèm theo đó là vô số chiêu trò khuyến mãi, chỉ cần bỏ 97\$ bạn sẽ thu được thứ trị giá gần 600\$. Tiếc cho anh Sonmez là, tuy anh quảng cáo khá thành công nhưng mà em nghèo quá, gấp 100 nghìn chắc em bỏ tiền ủng hộ anh rồi.

Một số bài học rút ra được

Có vài điều khá thú vị mình rút ra được sau khi dính cú marketing này của anh John:

- Đầu tiên, gửi vài cái mail úp mở để thu hút sự tò mò, chú ý của khách hàng. Nếu không có 2 email đầu tiên, chắc mình chả thèm click vào link giới thiệu đâu.
- Nội dung bài quảng cáo không tập trung vào bản thân mà tập trung vào chính khách hàng, vào những thứ họ nhận được. Các bạn nên học thử một khóa “Viết PR quảng cáo” để hiểu rõ hơn về cách viết này. Cá nhân mình thấy có một bạn viết quảng cáo học lập trình khá hay ở [đây](#), các bạn sinh viên hẵn sẽ rất hứng thú khi đọc.
- Tiếng tăm, tên tuổi của John Sonmez cũng khá quan trọng. Nếu là một thằng bá vơ nào nói nó học và dạy 1 ngôn ngữ trong 10 ngày thì mình không tin đâu, nhưng với anh John Sonmez thì khác, có người thật việc thật bằng chứng rõ ràng cơ mà. Các bạn thấy [thương hiệu bản thân](#) quan trọng chưa nào.
- Một trò nữa là “Tạo cảm giác gấp gáp, giá cả chỉ ưu đãi trong 6 ngày đầu tiên”, để dụ dỗ khách hàng mua sản phẩm, các bạn học marketing chắc không lạ gì trò này nhỉ.



Quay lại điều mình muốn nói ở đầu bài viết: Các bạn developer hãy thử tìm hiểu/học về marketing đi, nếu không bán được hàng thì cũng **bán thân được giá cao hơn**. Tác giả [blog codinghorror](#) cũng có một bài viết tương tự: <http://blog.codinghorror.com/the-one-thing-every-software-engineer-should-know/>

*If there was one thing I could teach every engineer, it would be **how to market**.*

Hai cuốn sách về marketing bản thân mà bạn nên tìm đọc thử là: *Whatever You Think, Think the Opposite* và *It's Not How Good You Are, It's How Good You Want to Be*. Bạn nào có sách hay hơn thì cứ giới thiệu nhé.

[Tutorial] Viết ứng dụng di động một cách dễ dàng với Ionic Framework

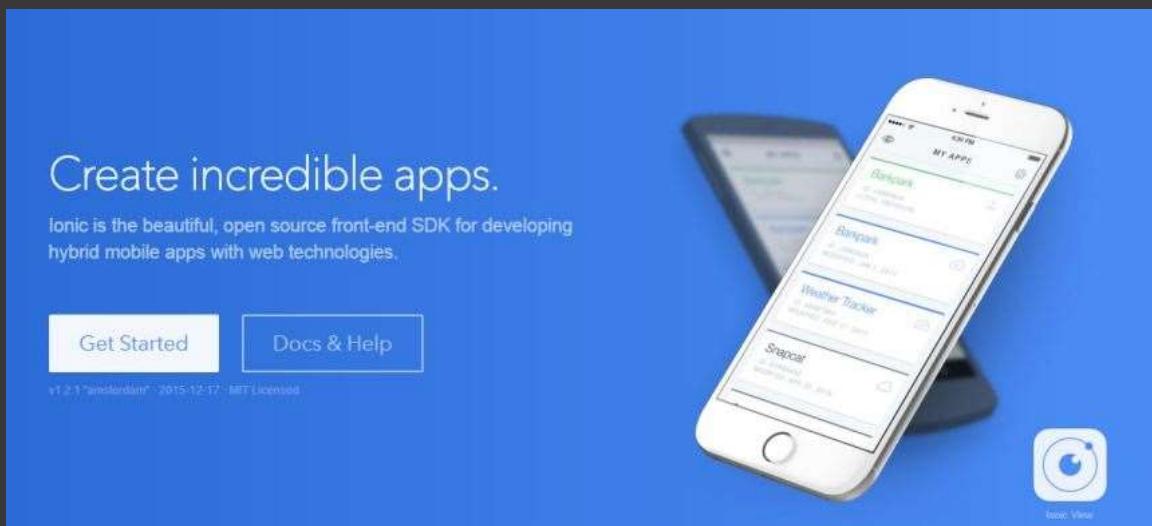
Posted on 12/01/2016 by Phạm Huy Hoàng

Nhu mình đã giới thiệu ở một số bài viết trước, [lập trình ứng dụng di động](#) đang là một lĩnh vực khá hot mà các lập trình viên nên quan tâm. Theo cách truyền thống (hoặc khi viết native app), bạn cần phải học Java, Objective-C, C# để viết ứng dụng cho Android, iOS, Windows-Phone.

Tuy nhiên, nếu viết ứng dụng theo hướng [hybrid app](#), sử dụng một số framework như Cordova, Phonegap, ... bạn có thể viết 1 ứng dụng hoàn toàn bằng **HTML, CSS và Javascript**, chạy được trên cả **iOS, Android và Window Phone**. Các bạn web developer có thể dễ dàng nhảy qua mảng mobile mà không cần tốn quá nhiều công sức để học ngôn ngữ mới.

Trong phạm vi bài viết này, mình sẽ hướng dẫn các bạn **tạo, viết và chạy ứng dụng** trực tiếp trên di động một cách nhanh và đơn giản với IonicFramework – một framework xây dựng dựa trên AngularJS và Cordova. Ưu điểm của IonicFramework là:

- Cài đặt rất nhanh và nhẹ. Thay vì phải tải IDE như Android Studio, Xcode cùng với bộ SDK nặng nề, ta chỉ cần tải NodeJS và dùng command prompt để cài ionic và cordova, rất nhanh và tiện lợi. Ta có thể dùng Notepad++ hoặc Sublime Text để code.
- **Chỉ cần biết HTML, CSS, JS và AngularJS.** Nếu muốn thêm 1 số chức năng như rung, camera, chỉ cần bỏ 5p để xem API ngCordova là làm được ngay.
- Tích hợp sẵn IonicViewer, cho phép ta **test app vừa viết ngay trên di động**, không cần cắm dây hay kết nối phức tạp (Việc debug thì còn hơi phức tạp đôi chút...). Kết hợp với Phonegap Build, ta có thể xuất file apk ngay và luôn.



Bắt tay vào làm thôi nào! Nếu chưa biết về AngularJS, bạn có thể google và tìm một số bài viết tiếng Việt, hoặc đọc tutorial trên blog của bạn mình để ứng

hộ:<https://codeaholicguy.wordpress.com/category/angularjs-cho-nguo-moi/>

1. Cài đặt Ionic

Đầu tiên, bạn vào <https://nodejs.org/> để tải nodeJS về và cài đặt, nhớ dùng bản 4 nhé, bản 5 chưa hỗ trợ Ionic.

A screenshot of the Node.js website. The top navigation bar includes links for HOME, ABOUT, DOWNLOADS, DOCS, FOUNDATION, GET INVOLVED, SECURITY, and NEWS. The main content area starts with a brief description of Node.js as a JavaScript runtime built on Chrome's V8 engine, using an event-driven, non-blocking I/O model. It highlights the large package ecosystem on npm. A green box at the top right urges users to "Important security releases, please update now!". Below this are download links for Windows (x64) in two versions: "v4.2.3 LTS" (Mature and Dependable) and "v5.3.0 Stable" (Latest Features). At the bottom of the page are links for "Other Downloads", "Changelog", and "API Docs".

Sau khi cài xong, bạn mở cửa sổ console lên, nhập vào:

```
npm install -g cordova ionic
```

Chờ khoảng 10p, việc cài đặt đã xong rồi đấy. Rất nhanh gọn lẹ phải không nào, chúng ta có thể bắt đầu viết ứng dụng rồi.

2. Tạo ứng dụng đầu tiên

Ở đây, mình sẽ tạo ứng dụng vào 1 folder có tên ionicApp. Để tạo ứng dụng, bạn cd vào thư mục, dùng câu lệnh phía dưới để tạo 1 ứng dụng có sẵn tabs. Chờ một chút, ứng dụng sẽ được tạo. Các bạn có thể đọc thêm ở đây: <http://ionicframework.com/getting-started/>.

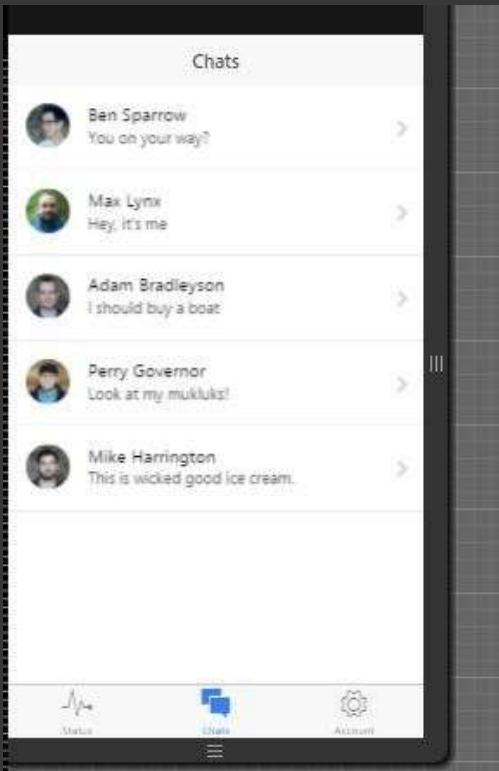
```
ionic start sampleApp tabs
```

```
C:\Users\Huy Hoang>e:  
E:>cd IonicApp  
E:>IonicApp> ionic start sampleApp tabs  
Creating Ionic app in folder E:\IonicApp\sampleApp based on tabs project  
Downloading: https://github.com/driftyco/ionic-app-base/archive/master.zip  
[=====] 100% 0.0s  
Downloading: https://github.com/driftyco/ionic-starter-tabs/archive/master.zip  
[=====] 100% 0.0s  
Updated the hooks directory to have execute permissions  
Update Config.xml  
Initializing cordova project  
  
Your Ionic project is ready to go! Some quick tips:  
  
* cd into your project: $ cd sampleApp  
  
* Setup this project to use Sass: ionic setup sass  
  
* Develop in the browser with live reload: ionic serve
```

Để chạy thử, ta chỉ cần cd vào thư mục sampleApp, gõ câu lệnh “*ionic serve*” để chạy thử ứng dụng trên nền web.

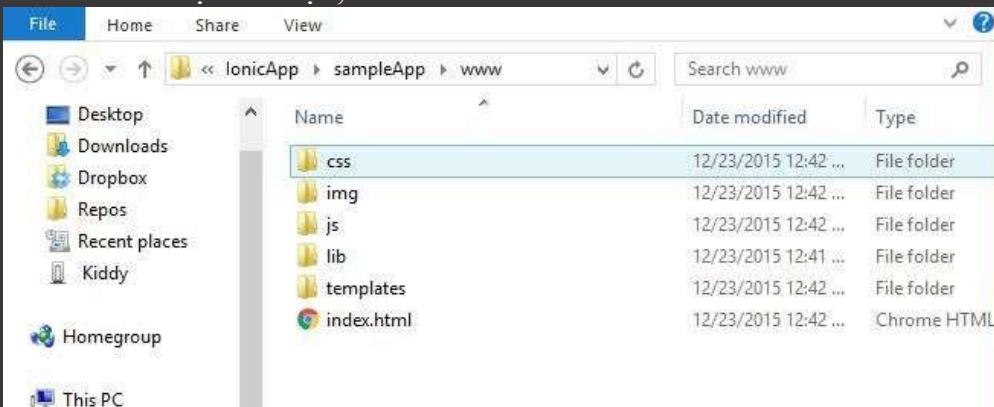
```
E:>IonicApp>cd sampleApp  
E:>IonicApp\sampleApp>ionic serve  
Running live reload server: undefined  
Watching: 0=www/**/*, 1=!www/lib/**/*  
Running dev server: http://localhost:8100  
Ionic server commands, enter:  
  restart or r to restart the client app from the root  
  goto or g and a url to have the app navigate to the given url  
  consolelogs or c to enable/disable console log output  
  serverlogs or s to enable/disable server log output  
  quit or q to shutdown the server and exit  
  
ionic $
```

Ta có thể thu nhỏ trình duyệt, hoặc dùng [Chrome Developer Tool](#) để xem thử ứng dụng hiển thị như thế nào trên mobile. Nhớ thử touch các tab và scroll lên xuống thử nhé.



3. Chính sửa ứng dụng

Toàn bộ code của ứng dụng sẽ nằm trong thư mục `sampleApp/www`. Trông quen quá nhỉ, chỉ có HTML, CSS, Javascript thôi. Mình khuyên các bạn nên dùng Sublime Text để mở toàn bộ thư mục, dễ chỉnh sửa hơn.



Do AngularJS sử dụng routing và template, một vài màn hình sẽ được lưu trong thư mục `templates`. Bạn thử chỉnh sửa file `tab-dash.html` trong thư mục `templates` và save lại nhé. Ionic hỗ trợ **live reload**, khi ta sửa một file, app sẽ tự refresh lại, không cần phải F5 gì cả.

4. Chạy ứng dụng trên di động

Tiếp tục ở cửa sổ console thần thánh, ta sử dụng câu lệnh sau để upload app lên server của ionic.

ionic upload

Nếu là lần đầu, bạn cần phải tạo một tài khoản và đăng nhập. Ở các lần sau ionic sẽ tự nhớ các thông tin này, bạn chỉ cần sửa app và bấm ionic upload thôi.

```
E:\IonicApp\sampleApp>ionic upload
No previous login existed. Attempting to log in now.

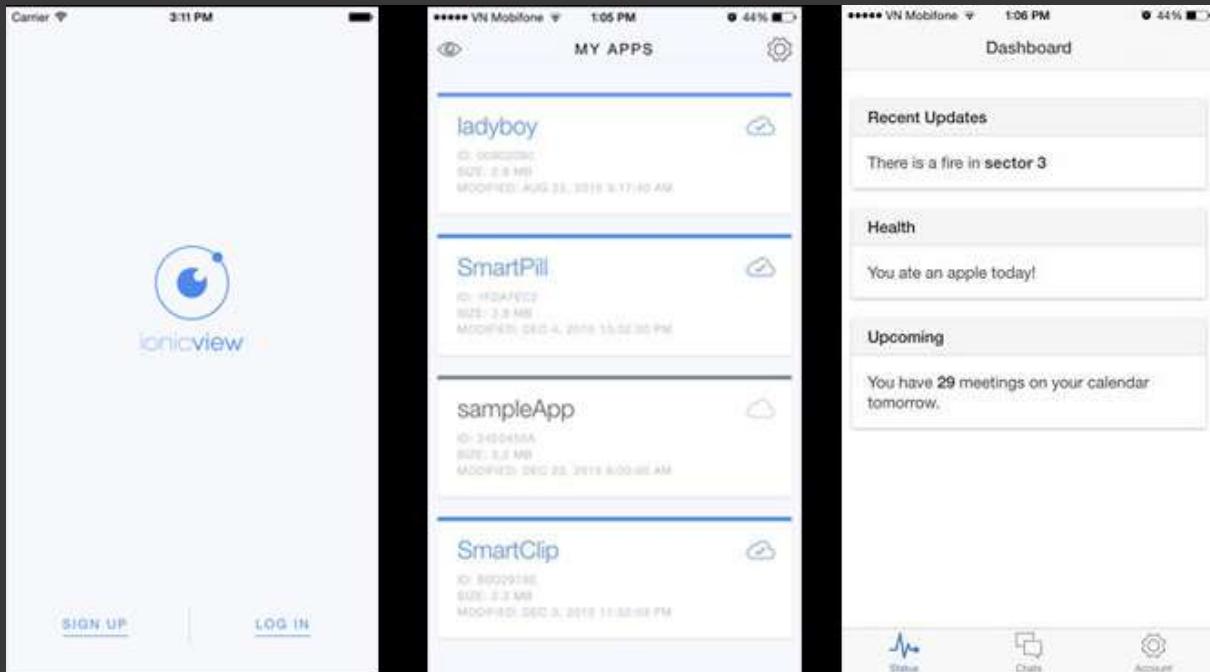
To continue, please login to your Ionic account.
Don't have one? Create a one at: https://apps.ionic.io/signup

Email: huyhoang8a5@gmail.com
Password:
Logged in! :)
Uploading app....
Saved app_id, writing to ionic.io.bundle.min.js...
Successfully uploaded (24e0455a)

Share your beautiful app with someone:
$ ionic share EMAIL

Saved api_key, writing to ionic.io.bundle.min.js...
```

Sau đó, bạn lên AppStore và tải IonicView về (IOS hay Android đều có cả). Cài đặt app và đăng nhập bằng tài khoản bạn vừa đăng ký nhé. Bạn sẽ thấy app của mình nằm ngay đó sau khi đăng nhập. Click vào và chịch... nhầm, nghịch thôi. Chúc mừng bạn đã hoàn thành bài hướng dẫn này.



Ngoài ra, nếu muốn build ứng dụng thành file apk để đăng lên mạng, đua bạn bè, bạn có 2 lựa chọn:

- Sử dụng *ionic build android*, cách này nhanh và đơn giản, nhưng bạn sẽ phải cài 1 đống SDK android khá nặng nè.
- Sử dụng Git để đưa project lên [github](#), sau đó sử dụng [Adobe Phonegap Build](#) để build online. Cách này hơi lòng vòng, nhưng không cần cài đặt làm nặng máy. Đây là một ứng dụng game nhỏ nhõ mà mình đã viết bằng Ionic, sử dụng Phonegap Build để build ra file apk:<https://dl.dropboxusercontent.com/u/46157401/ladyboy-debug.apk>

Bài viết tới đây là hết. Nếu có thắc mắc hay khó khăn gì, bạn có thể đặt câu hỏi trong phần comment nhé. Chúc các bạn may mắn.

Một button trị giá 300 triệu đô – Cái nhìn khác về UI và chức năng

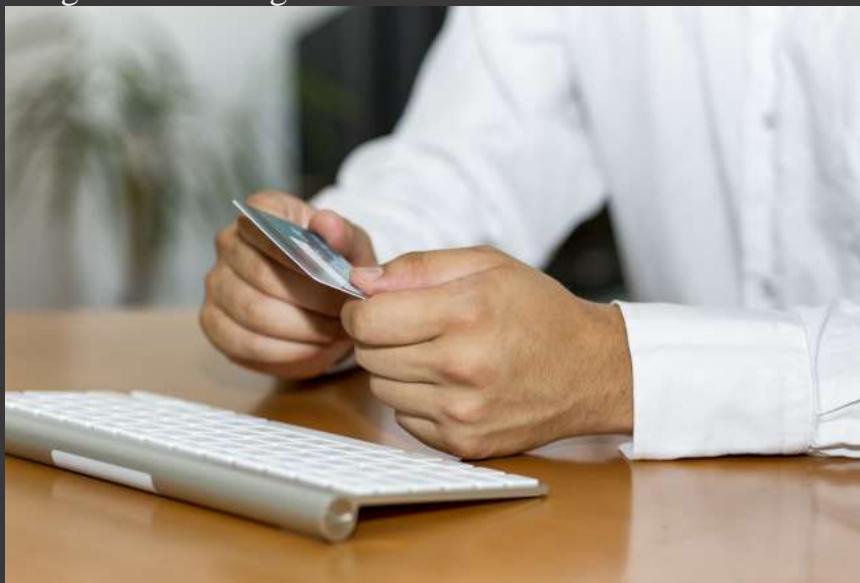
Posted on 07/01/2016 by Phạm Huy Hoàng

Ngày xưa ngày xưa, có một trang web bán hàng...

Bài viết đầu năm nên phải nói tiền trăm triệu cho nó hoành tráng. Đây là một chuyện nho nhỏ, về một button nho nhỏ và một số tiền... không nhỏ chút nào.

Mình đọc được chuyện này được trong cuốn Don't make me think – một cuốn sách khá hay về UI/UX. Ngày xưa ngày xưa, ở một đất nước nọ, có một trang web bán hàng... Chức năng cơ bản của một trang web bán hàng thì ai cũng biết: hiển thị hàng, cho hàng vào giỏ, và thanh toán.

Câu chuyện của chúng ta bắt đầu ở chức năng “Thanh toán”, khi người dùng đã cho hết hàng vào giỏ, một form nho nhỏ xinh xinh hiện ra, với 2 trường *username* và *password*, 2 nút *Login* và *Register*, một link *Quên mật khẩu*. Thế nhưng, chính cái form be bé xinh xinh này đã gây thiệt hại đến 300.000.000\$/năm cho trang web bán hàng.

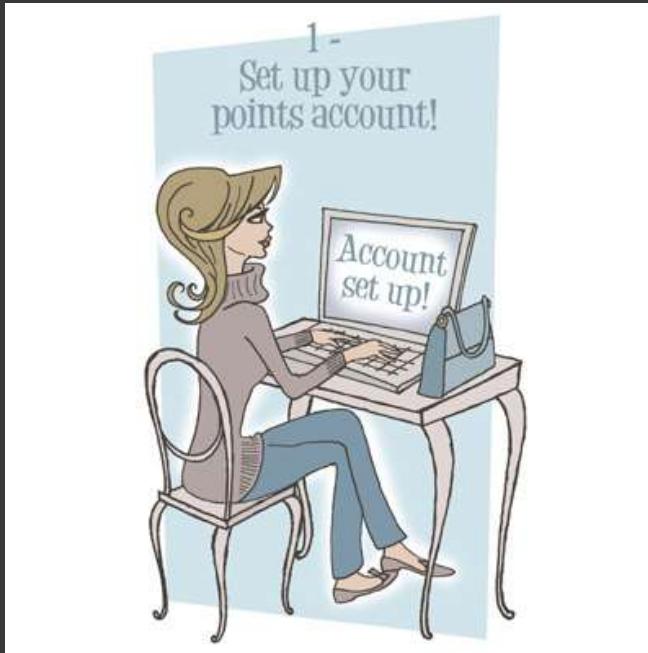


Vấn đề ở chỗ, người dùng phải **đăng nhập trước khi muốn thanh toán**. Đội lập trình nghĩ rằng “Chỉ cần đăng ký tài khoản, thông tin người dùng có thể được lưu lại, ở những lần sau họ không cần nhập địa chỉ, thông tin thanh toán nữa. Người dùng tiết kiệm được thời gian, web cũng khuyến khích được người dùng quay lại mua hàng, hai bên cùng có lợi còn gì?”.

Lũ lập trình viên luôn nghĩ rằng mình hiểu người dùng

Đội ngũ UI/UX đã làm một cuộc thử nghiệm, đưa tiền cho người dùng để họ thực hiện quá trình mua hàng – thanh toán. Đối với những khách hàng mới của trang web, họ phát hiện ra một điều: **người dùng rất ghét việc đăng ký**, với suy nghĩ “Mình muốn mua hàng, chứ không phải muốn đăng ký đăng ký cái gì cả”. Chưa kể, người dùng còn sợ bị mất thông tin cá nhân, bị gửi mail spam hộp thư.

Với những người dùng quay lại lần 2,3 – đối tượng mà developer nhắm tới, tình cảnh cũng chẳng khác. Họ không nhớ được username/mật khẩu của mình. Mặc dù chức năng “Quên password” vẫn hoạt động, đến tận 160.000 người dùng chức năng này mỗi ngày, **75% trong số đó không tiếp tục quá trình thanh toán** sau khi đã request mật khẩu.

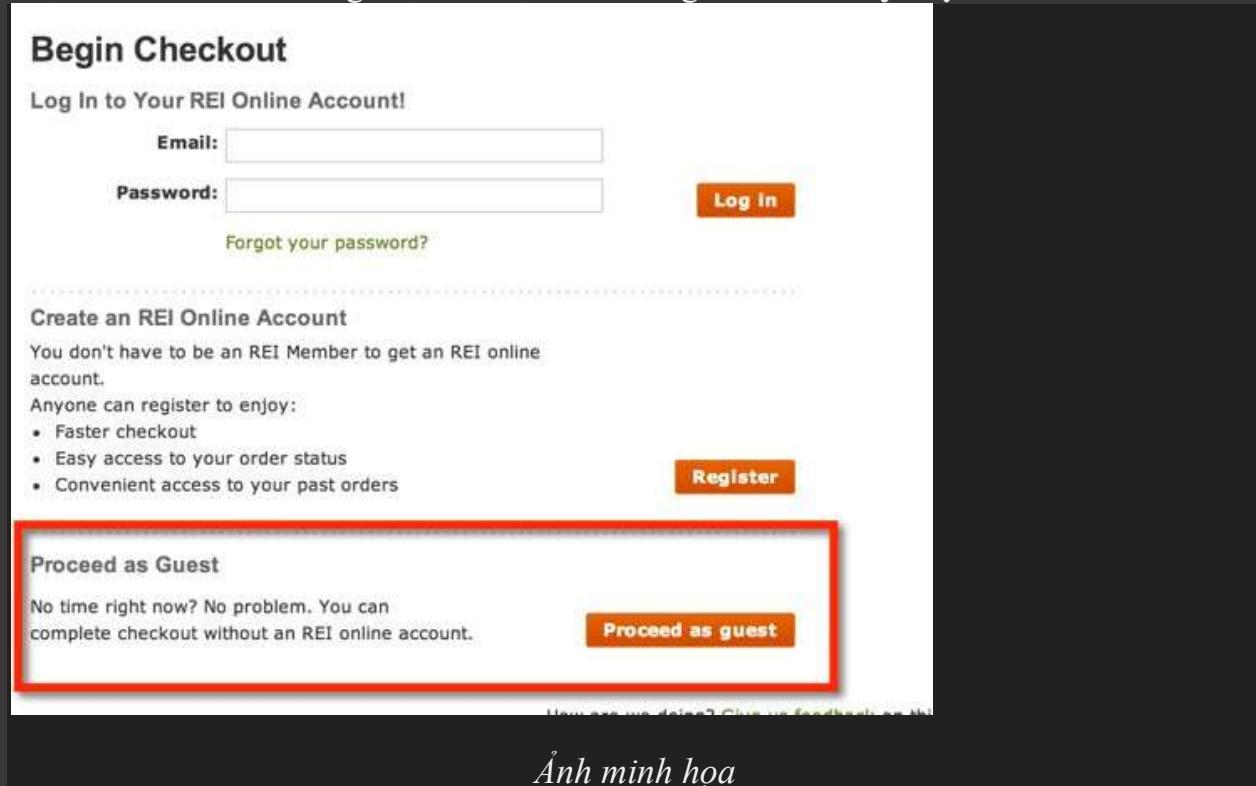


Chiếc form nho nhỏ xinh xinh kia, hóa ra lại là thứ ngăn cản người dùng mua hàng – rất nhiều người dùng. Thế mới biết, developer lúc nào cũng nghĩ mình hiểu được người dùng, nhưng thật ra không phải vậy.

Chiếc button trị giá 300 triệu đô la

Đội ngũ designer đã giải quyết vấn đề này một cách vô cùng đơn giản. Họ bỏ đi nút *Register*, thay vào đó bằng nút *Continue* và dòng chữ “Bạn không cần đăng ký, hãy bấm nút Continue để thanh toán. Để thanh toán nhanh hơn ở những lần sau, bạn có thể đăng ký một tài khoản vào lúc thanh toán”.

Kết quả: Lượng thanh toán của khách hàng tăng lên đến 45%. Sau tháng đầu tiên, doanh số tăng 15.000.000\$. Sau năm đầu tiên, thu nhập của web tăng đến tận 300 triệu đô la. Tất cả những việc mà họ đã làm là gì? Chỉ là **thay một button** mà thôi.



Một cái nhìn khác về UI/UX và chức năng

Một số bạn còn nhầm lẫn về hai khái niệm này, nên mình xin đưa ra một khái niệm tổng quát:

- **UI (User Interface) – Giao diện người dùng:** Đây là những gì người dùng nhìn thấy, tương tác được (Form, button, website).
- **UX (User Experience) – Trải nghiệm người dùng:** Đây là những gì người dùng trải nghiệm, bao gồm cảm xúc, suy nghĩ, quá trình. UI chỉ là một phần của UX. VD như khi bạn mua hàng trên tiki. Các trang web thanh toán, web mua hàng chính là UI. Quá trình mua hàng/nhận hàng, sự thoải mái khi thanh toán, sự hỗ trợ của Tiki với khách hàng chính là UX.

Trong câu chuyện trên, bằng cách *thay đổi UI* (Thay một button), họ đã trực tiếp *cải thiện UX* (Thay đổi trải nghiệm mua hàng, mua nhanh hơn, không cần đăng ký), từ đó làm tăng doanh số bán hàng.



Hẳn nhiều bạn lập trình viên vẫn còn suy nghĩ: Chức năng mới là quan trọng, còn mấy thứ ruồi bu như giao diện thì làm thế nào cũng được (Một phần chắc là do lúc mới học lập trình toàn phải viết chương trình Console, trước đây mình cũng vậy). Đây là một suy nghĩ hoàn toàn sai lầm.

Nếu chỉ viết vài tool đơn giản cho chính mình hoặc bạn bè xài thì đúng là giao diện không quan trọng. Nhưng nếu đã tạo ra sản phẩm, hướng tới người dùng, thì UI/UX là những thứ quan trọng nhất nhì, có khi hơn cả chức năng.

Nếu không tin, bạn hãy thử đọc **The Inmates Are Running the Asylum** – cuốn “thánh kinh” của dân thiết kế UI/UX, để hiểu rõ hơn về tầm quan trọng của hai thứ này. Hãy nhớ một điều mình muốn nói qua câu chuyện này: **Đôi khi chỉ một button nhỏ nhở lại có giá trị đến 300 triệu đô đấy.**

Các bạn có thể đọc thêm về câu chuyện “kinh điển” về UI/UX này ở đây:

- https://www.uie.com/articles/three_hund_million_button/
- <http://boingboing.net/2011/08/05/300-million-button-making-customers-create-logins-to-buy-cost-etailer-300myear.html>
- <http://www.fastcompany.com/1147825/300-million-continue-button>

Nhìn lại năm 2015, mừng blog thêm tuổi mới

Posted on 05/01/2016 by Phạm Huy Hoàng

Mình bắt đầu đặt bút ... nhảm, đặt phím viết bài đầu tiên cho blog vào ngày 31/12/2014. Hôm nay ngày 31/12/2015, mình viết bài này mừng blog tròn 1 năm tuổi. Thấy Facebook có chức năng “một năm nhìn lại”, mình cũng “nhìn lại” hoạt động của blog trong thời gian một năm vừa qua.

12 tháng – Những chuyện ngày xưa ngày xưa

Vào thuở mới viết blog, mình đã xác định hướng đi là viết về technical và “một số thứ khác nữa”. Thuở đó, kiến thức technical của mình còn hạn hẹp, còn “một số thứ khác” cũng chỉ là một khái niệm khá mơ hồ. Ở những bài đầu, mình tập trung viết về C#, javascript, cũng như một số kiến thức và định hướng trong ngành lập trình.

Mình tự đặt mục tiêu là viết 2 bài/tuần, đăng vào mỗi thứ 3 và thứ 5, ít nhất cũng phải hơn 1 năm mới bõ. Vì áp lực 2 bài/tuần, mình phải cố gắng trau dồi, rèn luyện thêm kiến thức để có cái mà viết. Nhờ viết blog mà kỹ năng viết và khả năng diễn đạt của mình được cải thiện khá nhiều.

Những hôm đầu, blog một ngày chỉ lèo tèo vài view, **cả tháng chỉ được chưa tới 500 view**. Cũng nhờ sự góp ý, ủng hộ mà cỗ vũ của các bạn mà blog đã dần vượt qua được các cột mốc 1000 view, post thứ 50, 100.000 view, và ngay ngày hôm nay thì **lượng view của blog đã lên tới con số 200.000, fanpage hơn 700 lượt like, khoảng 1-1k5 lượt view một ngày**.



Mặc dù mình đã cảm ơn **rất nhiều lần rồi**, hôm nay mình xin cảm ơn **thêm một lần nữa**. Các bạn nhớ đăng ký, like fan page và chia sẻ giới thiệu blog cho bạn bè nhé =))).

Cùng nhìn lại những cái nhất của blog

- **Bài viết được xem nhiều nhất:** [Mặt tối của ngành công nghiệp IT](#) với hơn 6k8 lượt xem. Theo sau là bài viết [C# là ngôn ngữ tuyệt vời nhất](#). Có vẻ dân tình thích tranh cãi với gây lộn, bài nào tên càng hot thì càng nhiều lượt xem và bình luận. Một bài viết cũng khá hay được nhiều bạn xem đó là: [Con đường phát triển sự nghiệp cho developer](#).
- **Bài viết được nhiều comment nhất:** [Thực trạng học lập trình của một số thanh niên hiện nay](#) do mình viết khi đang... say xỉn (Vừa viết cách đây không lâu nhưng được tận gần 50 comment và tranh cãi). Bài viết nhận nhiều gạch đá thứ hai chính là [C# là ngôn ngữ tuyệt vời nhất](#), âu cũng dễ hiểu với cách đặt tên gây sốc.
- **Ngày có nhiều view nhất:** 10.438 view vào ngày 20/11, trùng vào ngày nhà giáo Việt Nam. Hôm đấy mình đăng bài [Mặt tối của ngành IT](#). Tiêu đề gây sốc lại đi quảng cáo nhiều nơi nên view cao cũng là điều dễ hiểu.
- **Tháng có nhiều view nhất:** Hai tháng cuối năm 11 và 12 đang cạnh tranh vị trí này, với gần 55k view mỗi tháng. Chắc do số lượng người theo dõi tăng dần nên view cũng tăng theo.
- **Người comment nhiệt tình nhất:** Bạn PhucNguyen với 14 comment, tiếc là gần đây không thấy bạn comment gì nữa.

Hi vọng những kỉ lục này của blog sẽ được phá trong năm 2016.



Định hướng cho năm 2016

Do mình đã nghỉ việc ở [Aswig Solutions](#) vào cuối năm 2016 để đi [du học](#) ở trời Tây, mình cũng không bị áp lực công việc mấy mà có thể tập trung nhiều vào việc học và đọc sách. Tuy nhiên, do vào khoảng tháng 4 mình sẽ đi thực tập (khoảng 6 tháng) ở một công ty ở Anh theo chương trình học, mình quyết định sẽ không tập trung vào [một ngôn ngữ/công nghệ](#) cố định, không chuyên sâu mà tập trung mở rộng kiến thức.

Ngoài ra, trong năm nay mình sẽ trau dồi thêm kiến thức về [front-end](#) (Không chỉ là HTML/CSS/JavaScript mà còn lần sân luôn mây bạn designer, học Photoshop, InDesign, thiết kế UI và UX). Rành cả front-end và back-end thì quãng đi đâu cũng sống được cả. Do đó, blog sẽ có thêm một số bài viết về [thiết kế và UI/UX](#), tư duy sản phẩm. Những kiến thức này vẫn sẽ có ích cho bạn dù cho bạn có là back-end hay front-end developer, nhớ đón xem và ủng hộ blog nhé.



WordPress cũng khá hay, tự động tạo sẵn một báo cáo kha chi tiết và đầy đủ cho năm 2015. Các bạn có thể xem bản full ở đây: <https://toidicodedao.wordpress.com/2015/annual-report/>

Chuyên học tiếng Anh – Phần 1: Tôi đã học tiếng Anh như thế nào

Posted on 25/02/2016 by Phạm Huy Hoàng

Thấy cũng có nhiều bạn hỏi về cách học tiếng Anh và ôn thi của mình sẵn tiện mình viết bài chia sẻ luôn. Có 2 điều mình muốn nói rõ trước khi bắt đầu bài viết:

- Học là **một quá trình lâu dài**. Trừ khi bạn là siêu nhân hay thiên tài nhìn chữ là nhớ, còn lại thì học gì cũng **cần thời gian** để xây dựng nền móng mới giỏi dần được. Mình không phải thiên tài cũng không phải siêu nhân, do đó mình cũng phải học nhiều và học dần dần thì tiếng Anh mới khá được. Bạn nào sấp thimuốn tìm tips, cách học nhanh thì vui lòng tìm ở trang khác nha.
- **Mỗi người có một kinh nghiệm/cách học khác nhau**. Có người thích học tà tà mỗi ngày nửa tiếng, có người thích cày nhu trâu mỗi ngày 8 tiếng. Cách mình chia sẻ là **cách mình thấy phù hợp với bản thân mình**, các bạn thấy phần nào hợp với chính mình thì làm theo, đừng nghe và làm theo mình hết 100%. Nếu không thấy hiệu quả thì mình cũng không chịu trách nhiệm đâu.

Đây, mình nói rõ ràng từ đầu rồi nhé, không phải như quảng cáo của mấy cái lớp tiếng Anh “Làm theo phương pháp này 100% tiếng Anh bạn sẽ tiến bộ!” hoặc “Đã có 100 người thành công nhờ phương pháp này” v...v. Cách của mình chỉ giúp mình tiến bộ và có mình thành công thôi, nếu các bạn làm theo và thành công thì nhớ PM báo mình để mình cập nhật thêm nhé. Trong bài này, mình sẽ kể một chút về quá trình học tiếng Anh của mình.



Ngày xưa ngày xưa

Hồi lớp 10-11, tiếng Anh mình cũng chỉ thuộc hàng kha khá trong lớp chứ cũng không phải là nỗi trội hay xuất sắc gì mấy. Đối với mình thời đó, tiếng Anh chỉ là môn học và là công cụ hỗ trợ... chơi game. Công bằng mà nói, games là công cụ dạy tiếng Anh rất tốt. Nếu muốn, bạn có thể bỏ ra 2-3 tiếng mỗi ngày để chơi games, vừa giải trí vừa học tiếng Anh luôn thế.

Dĩ nhiên, games ở đây không phải là DotA, LoL, CS hay mấy trò webgame nhanh nhẹn trên mạng nhé. Thể loại game tốt nhất giúp tăng khả năng tiếng Anh là game nhập vai (JRPG của Nhật, RPG của châu Âu), bạn sẽ **quen với việc đọc chữ, đọc lời thoại**, xem quest để tìm cách làm. Một số game nhập vai khá hay bạn nên thử là: Series Final Fantasy 6-10, Series Tales of... Dòng Dragon Age rất hay từ gameplay cho tới nội dung nhưng tiếng Anh khá khó và phức tạp, bạn nào muốn thử thách bản thân cũng có thể thử. Nhớ chọn những game nhiều chữ nhiều lời thoại chứ đừng chơi Diablo, Torchlight hay Dungeon Siege nhe.

Thời đó mình còn tìm đọc hướng dẫn game tiếng Anh, nguyên file text dày đặc chữ, đọc nhiều cũng quen nên về sau thấy tiếng Anh nhiều mình không sợ nữa. Ngoài ra, chơi game online phiên bản quốc tế cũng làm khả năng đọc tăng đáng kể (Đọc thôi nhé, bạn nước ngoài giao tiếp trong game toàn dùng tiếng lóng, viết tắt, sai chính tả

v...v, không học được gì về giao tiếp đâu). Ngày xưa mình chưa Maple Story bản International còn dụ dỗ két hôn (trong game) với một bé gái 14-15 tuổi gì đó bên Phần Lan cơ :v.



Cuối năm cấp 3

Đến cuối năm lớp 12, do tìm một số bộ anime lên mình bị sa chân vào ma đạo, bắt đầu con đường cày cuốc anime và manga. Mình nhớ hồi đó là khoảng năm 2008-2009, trang vnsharing.net mới thành lập không lâu. Năm ngoái nó vừa sập vì nhiều lý do, tính ra mình cũng có nhiều bạn bè/kỉ niệm với nó, kể cũng buồn.

Thời đó, số lượng các nhóm sub anime/dịch truyện còn ít như lá mùa thu, không nhanh như bây giờ. Do đó, mình phải cắn răng mà mò mẫm đọc truyện/xem anime bằng tiếng Anh. Nghĩ lại cũng may chứ nếu hồi xưa có nhiều sub Việt như bây giờ chắc mình chả đụng tới tiếng Anh nữa. Theo kinh nghiệm của mình thì anime/manga cũng là một cách để học tiếng Anh khá tốt, vừa học vừa giải trí. Anime dễ xem hơn vì nhân vật nói từng câu, không phải nguyên 1 đồng chữ như trong manga. Ngôn ngữ trong anime/manga thì đủ thể loại, ma pháp học đường chính trị đủ cả. Các bạn nhập môn có thể tìm xem một số bộ đơn giản như: Naruto, One

Piece, Yotsuba, School Rumble, Aria, ... (Xem anime trong sáng chúa đừng lâm mấy bộ 18+ nhé, thiện ***tai thiện ***tai).

Nếu thích đọc manga bạn cũng có thể thử đọc tiếng Anh thay vì tiếng Việt. **Thay vì mỗi ngày vào blogtruyen, manga24h, bạn hãy vào mangapark, mangafox...** Vốn từ tiếng Anh của bạn sẽ lên dần dần mà không cần cố công rèn luyện học hành gì đâu. Nhiều khi đọc quen thì bạn đọc manga cả tiếng đồng hồ mà không biết luôn ấy chứ. Bạn cũng có thể bỏ manga và tablet và mang theo đọc. [iOS](#) có *iComic* là phần mềm đọc manga khá tốt, nếu bạn dùng [Android](#) thì có thể thử *Perfect Viewer*, ngày xưa mình dùng cả 2 cái này.



Lên đại học

Lên đại học, mình vẫn chơi game/xem anime/đọc manga như thường lệ. Do học FPT giáo trình toàn bộ bằng tiếng Anh, sách cuốn nào cuốn nấy dày cui như từ điển, mình cũng phải quen dần với việc đọc. Ở giai đoạn này, mình được giới thiệu [một số sách](#)khá hay (về lập trình và cuộc sống), nên mình bắt có thói quen đọc sách. Tuy nhiên hồi đó có khi cả tháng trời mình mới đọc xong một cuốn, không nhanh như bây giờ đâu.

Tất nhiên, nếu chưa quen, các bạn đừng nênhảy vào đọc nguyên một cuốn sách/truyện/tiểu thuyết bằng tiếng Anh, rất dễ ngập. Mình từng ảo tưởng sức mạnh đọc [Truyện kể Genji](#) (Tiểu thuyết hiện đại đầu tiên của nhân loại) và ôm đầu máu đầu hàng sau chương một. Các bạn có thể bắt đầu bằng thể loại Light novel, tiểu thuyết nhẹ nhàng theo phong cách anime/manga của Nhật, ngôn từ đơn giản và dễ hiểu. Thể

loại này rất hay, dễ ghiền, luyện tiếng Anh rất tốt, bạn nào muốn tìm hiểu thêm có thể vào đây nhe: <http://hako.re/forum/16-huong-dan/4830-light-novel-101-moi-dieu-can-biet.html>.



Một điều khá may mắn là sách technical cho dân lập trình chúng mình được viết bằng **văn phong đơn giản, súc tích dễ hiểu**, lại còn có nhiều code. Nếu đọc sách lập trình, bạn **không cần đọc hết từng câu từng chữ** mà chỉ cần nắm ý chính, hiểu code là được. Tuy nhiên, đọc ebook trên vi tính rất **mỗi mắt, dễ mất tập trung**, các bạn nên **bỏ sách vào di động hoặc tablet** để mang đi mà đọc. Đây là một số phần mềm nên dùng cho cách định dạng sách:

- Định dạng PRC + MOBI: Kindle. Kindle có thể cài kèm từ điển, gấp từ khó bạn chỉ cần giữ tay vào chữ sẽ thấy giải nghĩa ngay, rất tiện.
- Định dạng EPUB: Marvin
- Định dạng PDF: GoodReader

Mình đã dùng thử các phần mềm tương tự, chỉ giới thiệu cái tốt nhất cho các bạn đây. Nếu bạn không biết nên đọc sách gì, các bạn có thể xem một số bài giới thiệu trên blog của mình tại [đây](#), [đây](#) và [đây](#).

More energy was spent in the first few minutes of lift-off, in the first few miles of travel, than was used over the next several days to travel half a million miles.

Habits, too, have tremendous gravity pull—more than most people realize or would admit. Breaking deeply imbedded habitual tendencies such as procrastination, impatience, criticalness, or selfishness that violate basic principles of human effectiveness involves more than a little willpower. Note Highlight gravity pull, our freedom takes on a whole new dimension.

Like any natural force, gravity pull can work with us or against us. The gravity pull of some of our habits may currently be keeping us from going where we want to go. But it is also gravity pull that keeps our world together, that keeps the planets in their orbits and our universe in order. It is a powerful force, and if we use it effectively, we can use the gravity pull of habit to create the cohesiveness and order necessary to establish effectiveness in our lives.

"HABITS" DEFINED

For our purposes, we will define a habit as the intersection of knowledge, skill, and desire.

grav-i-ty /grāvētē/ n. 1 [PHYSICS] the force that attracts a body toward the center of the earth, or toward any other physical body having mass. For most purposes Newton's laws of gravity apply, with minor modifications to take the general theory of relativity into account.

[Google](#) [Wikipedia](#) [Full Definition](#)

Lời kết

Một điều mình đã từng chia sẻ nhiều lần trong các bài viết trước, đó là sự quan trọng của thói quen. Thói quen tốt sẽ giúp bạn dần dần đạt được điều mình muốn. Với mình, đó là thói quen đọc sách, viết blog, chơi game đọc truyện. Nếu muốn tăng khả năng tiếng Anh, ngoại trừ việc học, bạn có thể tập cho mình một vài thói quen giải trí bằng tiếng Anh, vừa vui vừa bõ.

Nếu các bạn có thắc mắc là: Ô sao toàn luyện Reading vậy, còn mấy cái như *Writing* hoặc *Speaking* thì sao???. Như đã chia sẻ, đây là **cách học của mình**, mình vừa học vừa chơi theo sở thích nên Reading và Listening kha khá tốt thôi. Writing và Speaking mình cũng không giỏi lắm nên không chia sẻ được gì mấy. Các bạn có thể thoải mái chia sẻ kiến thức/kinh nghiệm học tiếng Anh của mình trong phần comment nhe. Ở phần sau, mình sẽ chia sẻ một số kinh nghiệm về quá trình học ôn thi TOEIC, các bạn nhớ đón đọc nhé.

Chuyên học tiếng Anh – Phần 2: Tôi đã đạt TOEIC 945 như thế nào

Posted on 03/03/2016 by Phạm Huy Hoàng

Với nhiều bạn sinh viên, TOEIC là một kì thi khá quan trọng, vì nhiều trường đại học đòi hỏi tấm bằng TOEIC trên 400-600 điểm mới cấp bằng. Nối tiếp phần trước, trong bài viết này, mình sẽ chia sẻ lại một số kinh nghiệm quá trình ôn tập, học và thi TOEIC. Bài viết này ngắn thôi:

Mình ôn tập, học mất 2 tháng, sau đó bước vào phòng thi và làm bài. Sau 2 tiếng mình ra khỏi phòng, làm bài dư 15 phút. Mình về nhà chờ 1 tháng, sau đó quay lại trung tâm nhận kết quả. Chấm hết...

A detailed description of the IELTS Test Report Form: The form is titled 'INTERNATIONAL ENGLISH LANGUAGE TESTING SYSTEM Test Report Form ACADEMIC'. It contains fields for Centre Number (VN101), Date (28/FEB/2015), Candidate Number (011300), and Candidate Details (Family Name: PHAM, First Name: HUY HOANG, Candidate ID: 24123HUY). The form also includes fields for Date of Birth (04/03/1992), Sex (M), Schema Code (Private Candidate), Country or Region of Origin (VIETNAM), Country of Nationality (VIETNAM), and First Language (VIETNAMESE). The 'Test Results' section shows scores for Listening (6.0), Reading (6.2), Writing (6.5), Speaking (6.2), and Overall Band Score (7.5). The form includes an 'Administrator Comments' section with a signature, a 'Centre stamp' (IDP VIETNAM), a 'Validation stamp' (IELTS), and a 'Test Report Form Number' (1454010305PHAH01A).

Đùa các bạn tí áy mà, các bạn kéo xuống dưới để xem tiếp bài viết nhé.

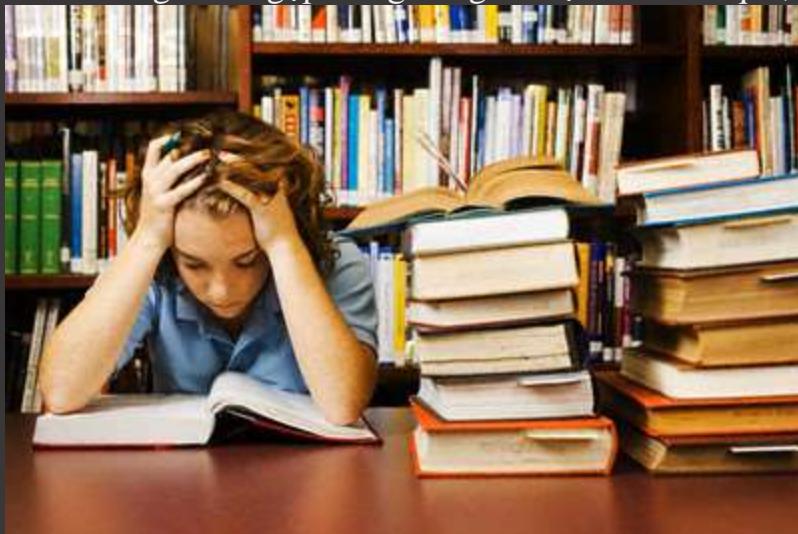
Chuẩn bị tinh thần và tài liệu

Một điều cần cảnh báo trước cho các bạn là: **TOEIC rất dễ**. Nhiều bạn cứ tưởng nó ghê gớm lắm chứ tới lúc bắt đầu ôn rồi mới thấy nó **dễ không tả được**, chắc chỉ khó hơn **đề thi Anh Văn tốt nghiệp** một chút thôi. Bạn bè mình nhiều đứa Anh Văn không giỏi nhưng tự ôn 2-3 tháng thi toàn được 7-800 điểm trở lên cả. Vì vậy nên các bạn nên bỏ tư tưởng sợ hãi đi nhé.

Ngoài ra, nội dung TOEIC là tiếng Anh thường nhật, công sở, chỉ có thể dùng khi đi xin việc được. Nếu bạn có ý định du học thì tập trung học và thi IELTS chứ đừng lấy bằng TOEIC làm gì nhé (Hồi đó mình rảnh nên thi cho vui thôi).

Thói quen của mình là **chuẩn bị đầy đủ các tài liệu và lên lịch** rồi mới bắt đầu học và ôn tập, không phải học tới đâu tìm tới đó. Các bạn tự học, tự ôn thi hay gấp một trong 2 vấn đề sau:

1. Không biết Google nên không biết nên học ôn thé nào/chuẩn bị tài liệu gì.
2. Biết Google nên tìm được một đống lời khuyên về học/ôn thi TOEIC, cuối cùng chết ngopalong trong đống tài liệu vì nhiều quá, không biết học cái nào.



Với trường hợp 1, các bạn có thể hỏi ý kiến/kinh nghiệm từ người thân, bạn bè hoặc... ra trung tâm học. Nói gì thì nói, tự học đòi hỏi bạn phải kiên trì và chịu khó xây dựng thói quen, khá là vất vả. Ngoài trung tâm họ sẽ có một kì thi nhỏ nhằm đánh giá trình độ của bạn, từ đó tư vấn lộ trình và lớp học phù hợp. Mình không học ngoài trung tâm nên không giới thiệu được trung tâm nào tốt đâu nhe.

Đa phần bạn bè của mình sa vào trường hợp 2, bỏ cả mấy ngày trời tải về đú thứ sách vở, tư liệu cả vài GB, sau đó tẩu hỏa nhập ma vì ... nhiều quá, chả biết học thế nào, bắt đầu từ đâu. Kinh nghiệm của bản thân mình là: Bạn không cần quá nhiều sách ôn tập, chỉ cần **đủ và chất lượng** thôi. Để chuẩn bị cho kì thi TOEIC mình chỉ đọc **đúng 4 cuốn sách dưới đây**.

- Starter TOEIC
- Developing Skills for TOEIC Test
- Toeic Analyst
- Target TOEIC

Vốn mình định để các bạn tự tìm link nhưng thôi hôm nay đang vui, share link luôn:[Download](#). Bạn nào down xong nhớ nhìn qua bên tay phải, bấm nút like cái [fanpage](#) của mình cái nào :P.

Giới thiệu nội dung

Tác giả là một người đam mê văn hóa Nhật. Trên một chuyến tàu điện, anh thấy một chàng trai đang hối hả sửa lại những slide chỉ chít chữ của mình. Nhìn xuống hộp bentou đang ăn dở, anh chợt này ra ý nghĩ “*Tại sao ta không tạo ra những slide đơn giản, thanh nhã, tinh tế như những hộp bentou*”. Đó là nguồn cảm hứng để anh áp dụng **chất Thiến** vào việc thuyết trình, thiết kế slide.

Theo thói quen thông thường, chúng ta hay nhét rất nhiều thông tin, số liệu vào slide. Hậu quả là slide dày đặc chỉ chít chữ, làm rối mắt khán giả. Sách giải thích rõ sai lầm cơ bản này:

- Slide là thứ khán giả nhìn và chú ý tới.
- Thông tin, số liệu sẽ được in ra dưới dạng handout và phát cho khán giả đọc, đừng nhét chúng vào slide.



Tôi đi code dạo
41 lượt thích

Thích Trang Chia sẻ

Hãy là người đầu tiên trong số bạn bè của bạn thích nội dung này!

Follow Blog

Hãy nhấn nút "Theo dõi" để nhận thông báo qua email khi có bài viết mới!

Join 109 other followers

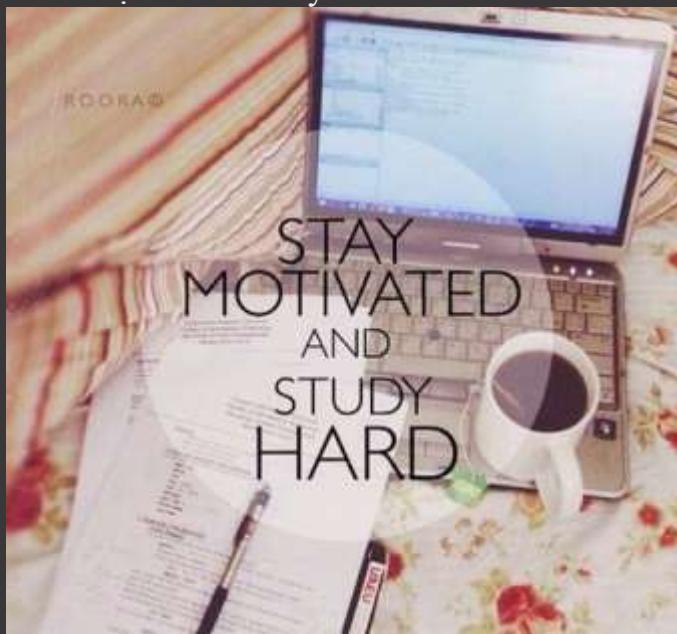
Enter your email address

Ôn thi TOEIC

Bạn chỉ bỏ thời gian ra khoảng **1 tiếng rưỡi – 2 tiếng** mỗi ngày trong vòng 2 tháng là xong ngay. Luyện thi TOEIC hay IELTS thì đều phải trải qua 3 giai đoạn.

- Giai đoạn 1: Ôn luyện lại kĩ năng tiếng Anh cơ bản nói chung như: Văn phạm, từ vựng, ngữ pháp, kĩ năng nghe đọc. Phần này thì mình không ôn nên không có tài liệu.
- Giai đoạn 2: Làm quen với cấu trúc đề TOEIC, các dạng câu hỏi, những điều cần lưu ý. Trong tài liệu mình chia sẻ là 3 cuốn: Starter TOEIC, Developing Skills for TOEIC Test, Toeic Analyst.
- Giai đoạn 3: Thi thử, tính thời gian và làm quen với đề. Trong cuốn Target TOEIC mình share có 6 đề và lời giải.

Sau 3 giai đoạn ôn luyện này, bạn đã có đủ khả năng để bước vào phòng thi và làm bài rồi. Nếu còn thời gian rảnh, bạn có thể xem thêm một số tips và trick để tăng điểm số. Tuy nhiên các bạn nên nhớ một điều là: **Khả năng tiếng Anh mới là **thứ quan trọng nhất giúp bạn đạt điểm cao****, những tips trick này chỉ giúp bạn không mất điểm một cách vô duyên thôi.



Lời kết

Quá trình thi cũng không có điều gì đáng nói. Bạn tới trung tâm đăng ký, chọn ngày thi và đóng tiền. Tới hôm thi nhớ cầm theo CMND/passport rồi vào phòng thi là được. Chất lượng âm thanh khá ổn, mỗi người có một tai nghe riêng nên đừng lo về phần Listening. Một bài viết về những điều cần lưu ý, kinh nghiệm rút ra khi làm bài có thể tìm được khá dễ dàng trên mạng. Mình không chuyên về tiếng Anh bằng người ta nên không viết lại nhé.

Như đã nói ở đầu bài viết, **TOEIC rất dễ, còn IELTS nǎm ở một đẳng cấp cao và khó hơn TOEIC nhiều lắm**, do đó quá trình ôn thi IELTS cũng lâu và lǎm gian truân hơn nhiều. Ở phần 3, mình sẽ chia sẻ về quá trình ôn và thi IELTS “đầy mồ hôi xương máu” này. Các bạn cứ thoải mái chia sẻ kinh nghiệm/cách luyện thi của mình trong topic này nhé.

Chuyên học tiếng Anh – Phần 3: Tôi đã đạt IELTS 7.5 như thế nào

Posted on 10/03/2016 by Phạm Huy Hoàng

Ở bài trước, mình đã chia sẻ một số kinh nghiệm hoc thi TOEIC. Trong bài viết này, mình sẽ chia sẻ một số kinh nghiệm học thi IELTS – một kì thi toàn diện và khó hơn TOEIC rất nhiều.

The image shows two versions of an IELTS Test Report Form. The left form is for the 'LISTENING AND READING INSTITUTIONAL PROGRAM SCORE REPORT' and the right form is for the 'INTERNATIONAL ENGLISH LANGUAGE TESTING SYSTEM Test Report Form'. Both forms are for the Academic module.

Candidate Details:

- Name: Phạm Huy Hoàng
- Date of Birth: 04/03/1992
- Sex (M/F): M
- Schema Code: Private Candidate
- Centre Number: VN101
- Date: 28/FEB/2015
- Candidate Number: 0110001
- Family Name: PHAM
- First Name: HUY HOANG
- Candidate ID: 241239101
- Country or Region of Origin: VIET NAM
- Nationality: VIETNAMESE
- First Language: VIETNAMESE

Test Results:

Listening: 6.5	Reading: 6.5	Writing: 6.5	Speaking: 6.5	Overall Band Score: 7.5
----------------	--------------	--------------	---------------	-------------------------

Administrative Comments:

[A large empty box for administrative comments, signed with a handwritten signature.]

Logos and Stamps:

- British Council logo
- idp logo
- Cambridge English Language Assessment logo
- IELTS logo

The validity of this IELTS Test Report Form can be verified online by recognized organizations at <http://www.idtac.org.uk>.

Như đã nói ở bài đầu, cách học của mình có thể sẽ phù hợp với bạn này nhưng không phù hợp với bạn khác. Các bạn nên chắt lọc, áp dụng những thứ mà bản thân dùng được chứ đừng làm theo 100% nhé.

Sơ lược về IELTS

IELTS là một chứng chỉ tiếng Anh. Nếu muốn du học, bạn cần chứng chỉ IELTS trên 6.5 (Một số trường nhận có IELTS và TOEFL, mình không thi TOEFL nên không rõ).

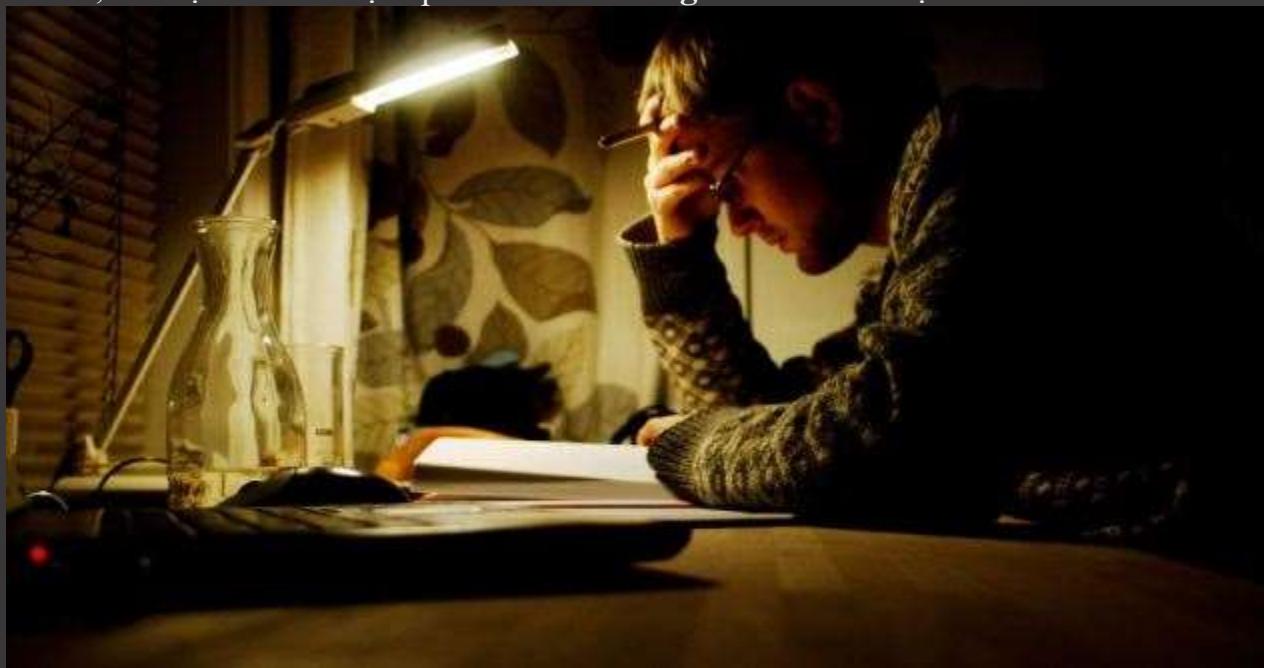
Vì IELTS có 2 dạng là General và Academic, dạng Academic tập trung nhiều hơn vào từ vựng chuyên ngành và học tập nên khó hơn. Về cấu trúc đề, lệ phí, thời gian thi... các bạn [chiều khó google](#) tìm hiểu nhé.

Chuẩn bị tinh thần và tài liệu

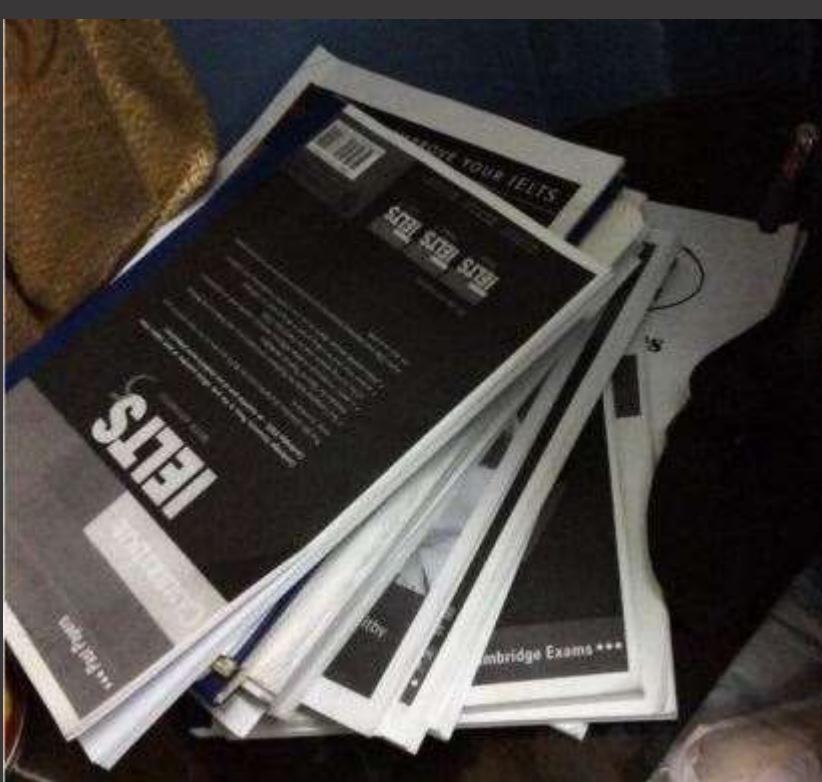
Ở bài trước, mình đã chia sẻ với các bạn rằng **TOEIC rất dễ**, đừng lo, cứ thả lỏng tinh thần, yên tâm mà học. Tuy rất tiếc, mình phải nói điều ngược lại so với IELTS. **IELTS khá khó**, nằm ở một đẳng cấp cao hơn nhiều so với TOEIC vì các lý do sau:

- IELTS có cả *nghe, nói, đọc, viết* trong khi TOEIC chỉ có *nghe, đọc*. Ôn thi IELTS phải ôn cả 4 kỹ năng.
- Vốn từ của TOEIC hầu hết tiếng Anh công sở, trong khi IELTS lại tập trung vào *Academic English*. Những từ dùng trong academic khá hiếm gặp và khó nhớ.
- TOEIC chỉ có trắc nghiệm, còn IELTS bạn phải trả lời kiểu tự luận, điền từ, sắp xếp. Điền từ mà chia sai thì hoặc số ít/số nhiều sẽ không có điểm.

Vì độ khó của IELTS cao hơn TOEIC nên **thời gian ôn luyện và học phí của các trung tâm cũng cao hơn**, 1 tháng chắc cũng tầm 1-2 triệu gì ấy. Thuở xưa mình ra APA, ACET hỏi, khóa nào cũng **17-25 triệu**, mắc quá nên mình đành tự học ở nhà. Do đó, các bạn cần xác định phải **kiên trì và nghiêm túc** khi học thi IELTS.



Ban đầu, thằng bạn quăng cho mình gần 30GB tài liệu ôn thi IELTS. Nhiều bạn khi ôn thi cũng tải về vài chục GB tài liệu rồi để đó, vì thấy nhiều nên... ngại. Kinh nghiệm của mình là các bạn chỉ cần lọc ra vài cuốn hay nhất, sau đó ôn luyện từ từ là được. Tài liệu ôn thi IELTS của mình cũng khoảng chục cuốn, ôn trong hơn 3 tháng, các bạn có thể tải về [ở đây](#) (Tải xong nhớ nhìn qua bên phải, bấm like [fanpage](#) nha). Nhớ in ra và đọc dần nhé, sách dạng này nếu đọc ebook sẽ mệt với mỏi mắt lắm. Ngày xưa mình đi in cả chục cuốn, tổng tiền chắc cũng gần 4-500 nghìn gì ấy, vẫn rẻ hơn học 1 buổi trung tâm. Đống sách ngày xưa của mình đây:



Ôn thi IELTS

Nhu đã chia sẻ ở bài trước, việc ôn thi của mình chia làm 3 giai đoạn: Ôn kiến thức nền tảng, luyện kỹ năng thi, thi thử. Các sách mình nói đều có trong link phía trên rồi nhé.

Giai đoạn 1: Ôn kiến thức nền tảng

3 tháng mình ôn thi IELTS bao gồm giai đoạn 2 và 3. Giai đoạn 1 này kéo dài lâu hơn, cũng khoảng 3 tháng, từ lúc mình xác định sẽ thi IELTS. Giai đoạn này chủ yếu dùng để luyện kỹ năng tiếng Anh:

- Mình dành khoảng 1 tháng lên [VOA Learning English](#) để tập nghe lại **và viết ra**. VOA nói chậm nên dễ nghe hiểu, tuy nhiên bạn phải tập nghe xong 1 câu, stop và viết lại câu đó. Ban đầu việc này khá khó, về sau mình quen dần. 1 ngày bạn chỉ cần bỏ ra khoảng 1 tiếng, nghe và viết cỡ 3-4 videos trên VOA là được.
- Sau khi đã quen, ta có thể lên [TED](#) để nghe các bài nói. Các bạn nên chọn những bài nói dài 10-15 phút, nghe 1 lần không phụ đề và đoán, sau đó nghe lại có phụ đề để xác minh lại. Từ vựng các bài nói trên TED khá khó, đôi khi người nói lại không phải giọng chuẩn, tập nghe những bài này sẽ giúp khả năng nghe tiếng Anh được cải thiện.
- Rèn lại từ vựng và phát âm: cuốn **Cambridge Vocabulary for IELTS** khá đú từ vựng Academic English và mấy cuốn **English Pronunciation in Use** dạy cách phát âm.

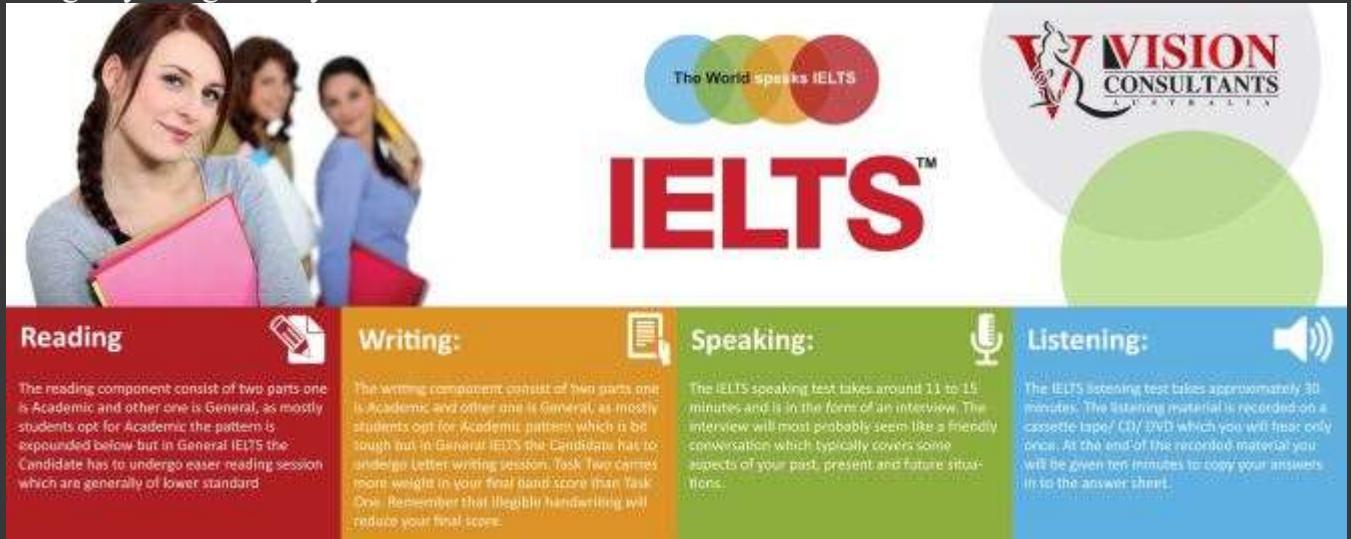


Giai đoạn 2: Luyện kỹ năng thi

Ở giai đoạn này, bạn sẽ tập làm quen mới cấu trúc đề, những dạng câu hỏi thường gặp, luyện kỹ năng thi:

- Lần lượt học 2 cuốn **IELTS Graduation Student Book** và **Study Skill**. Hai cuốn này giới thiệu khá đầy đủ các dạng câu hỏi thường gặp khi thi IELTS, có bài tập để ôn luyện.
- Sau khi học xong 2 cuốn này, chuyển qua 3 cuốn **Improve your IELTS Listening and Speaking, Reading, Writing**.
- Phần Writing là phần khó nhất với nhiều bạn (Xem thi tiếng Anh tốt nghiệp để dễ vậy mà còn nhiều người bỏ trống là biết). Với mình nói cũng khá khó nên mình có chuẩn bị thêm 1 số tài liệu trong folder **Writing Simon** và **SlidePDF by NgocBach**. Do tự học nên chúng ta có phần khá thiệt thòi, không có người feedback writing hay luyện speaking cùng.

Ngoài sách ra, các bạn nên lên trang [ielts-simon](#) để đọc và làm bài tập. Tác giả Simon là người ra đề/chấm thi IELTS, các lời khuyên ông đưa ra rất hữu ích, bài tập trên trang này cũng rất hay.



Giai đoạn 3: Thi thử

Sau khi ôn luyện đầy đủ, các bạn có thể làm thử đề thi IELTS trong folder **Cambridge IELTS Course and Practice Test**. Mấy quyển này thì ai ôn IELTS cũng biết, bạn có thể ra nhà sách mua 90k kèm đĩa, hoặc tự in ebook thì rẻ hon. Nếu rảnh, có thể xem một số tip and trick để kiểm điểm. Trong folder **Ránh Ránh đọc** có 3 cuốn dạng này, nội dung cũng khá hay, các bạn có thể xem nhanh 1,2 tuần trước khi thi.

	Common Mistakes at IELTS Intermediate.pdf	16/4/2014 3:34 PM
	IELTS Target band 7.pdf	16/11/2008 8:22 AM
	TIPS for IELTS by Sam McCarter [ani...hshumi].pdf	13/9/2012 10:26 AM

Lời kết

Nếu cảm thấy lười không có đồng lực học tập, các bạn hãy đi đăng kí thi trước. Lệ phí thi IELTS là **4 triệu rưỡi**, đóng tiền xong đảm bảo bạn sẽ cảm đầu vào học, không dám nghỉ vì xót tiền cho mà xem. Ngày xưa mình đăng kí thi vào cuối tháng 2, xong rồi học trâu bò cả Tết luôn. Chúc các bạn may mắn và đạt điểm cao. Bạn nào có tài liệu/kinh nghiệm gì hay muốn chia sẻ thì cứ comment nhé.

Chuyện đầu năm – Lần đầu đi phỏng vấn xin việc nơi đất khách quê người (Phần 1)

Posted on 16/02/2016 by Phạm Huy Hoàng

Năm nay Tết ta được nghỉ khá nhiều, tận 9 ngày. Do thấy số lượng người xem cũng ít nên blog cũng tạm “nghỉ Tết” cho các bạn đọc có thời gian giải lao, vui chơi quây quần bên người thân và bạn bè. Mình vẫn đang ở nước ngoài nên chả có Tết nhất gì, chỉ có 1 bữa ăn nho nhỏ với các bạn VN, may là gặp được 1 vài bé SV năm nhất khá xinh.

Sau hai tuần nghỉ Tết, chắc súc ý sau 10 ngày nghỉ vẫn còn nên bài đầu tiên này mình sẽ không viết về technical nhức đầu mà xin kể một câu chuyện nho nhỏ đầu năm mới. Tựa đề khác của câu chuyện là “Tôi đã đi code dạo ở nước ngoài như thế nào”. Đây là câu chuyện về quá trình xin việc (xin đi code dạo), cũng như lần đầu mình đi phỏng vấn nơi xứ la quê người.



Tác giả là thanh niên sơ mi xanh hàng trên cùng thứ tư từ trái vào (ảnh chắc đã qua bàn tay chỉnh sửa nên hơi hư cấu)

Duyên trời đưa đầy

Mình qua UK học Computer Science từ hồi tháng 9, trừ 1 tháng nghỉ hè ra thì cũng được hơn 4 tháng rồi. Ở giai đoạn đầu, do phải tập làm quen với cuộc sống và việc

học nên mình không có thời gian rảnh để đi làm thêm (Vả lại có [hoc bong 500](#) cũ cũng đủ tiền ăn tiền nhà rồi, cũng không cần xin gia đình gửi tiền qua). Qua học kì mới, thấy các môn có vẻ nhẹ, ít bài tập nên mình cũng khá rảnh rồi. Đang định dứt vài series dang dở trên [pluralsight](#) thì nghe thằng bạn da đen cùng lớp nói “Phòng IT của trường (Information Systems Services), chõ tao thực tập đang tuyển [developer](#) hay sao ấy, mà đăng ký thử đi, mà apply thì chắc là đậu”.

Nói về anh da đen này một chút (anh bên phải trong hình dưới). Cha này cũng học Computer Science nhưng mà code không giỏi, lại mất căn bản Java. Do duyên trời đưa đẩy, ngồi cạnh mình trong phòng Lab môn Distributed System, lại còn chung nhóm với môn Advanced HCI. Chắc do thấy mình làm nhóm thì toàn **hở báo chỉ đạo phân việc**, làm bài Lab thì lúc nào cũng **hở báo hỏi giáo viên** nên anh có vẻ nể mình lắm. Thấy tội tội nên lâu lâu nhò hướng dẫn mình cũng chịu khó giảng, có điều chắc ảnh không có kinh nghiệm code.



UX Developer là cái chi chi???

Quay lại chuyện xin việc. Mình lên trang rao việc xem thử thì thấy vị trí tuyển dụng là UX developer (part-time). Nghe cái title lạ hoắc lạ huo, xem đến phần yêu cầu công việc thì

Essential

1. *Working towards a degree (or equivalent) in a computer related discipline*
2. *Experience of programming in HTML CSS3 / LESS and JQuery in a range of scenarios*
3. *Experience of development in C# Application Form*
4. *Ability to work within a team, and under own initiative*
5. *Experience of achieving clear goals*
6. *Ability to plan, organise and solve complex problems*

Desirable

1. *Experience in Adobe Creative suite (Photoshop, Illustrator etc.), Sketch, Invision or similar prototype tool*
2. *Experience with UX modelling techniques such as case diagrams, use-case descriptions*
3. *A good understanding of website design and responsive UI and common frameworks, Foundation, Bootstrap*

Hóa ra là tuyển người **làm UI/UX**. Chắc là ông trời biết mình từ đâu năm đến giờ mình tập trung vào mảng front-end đây mà. Mình đã từng chém ở bài trước là rành cả front-end và back-end thì quăng đi đâu cũng sóng được cá, không ngờ linh nghiệm thật.

Nộp CV và chờ mòn mỏi

Khô thay, bên này xin việc nó không chỉ cần CV mà còn cần cả thư xin việc (cover letter) đính kèm. Trước giờ mình có viết cái cover letter này bao giờ đâu, thế là phải lên google tìm một mẫu hay hay, ché lại thành của mình rồi gửi. Chờ 1 ngày, 2 ngày, 1 tuần rồi mà vẫn chưa thấy đâu. Mình tưởng là rót rồi nên chửi thầm “Mẹ, mình từng viết series hướng dẫn xin việc – phỏng vấn mà giờ lại trượt ngay vòng gửi CV, nhục vãi”. May thay, sau 3-4 ngày, cuối cùng mình cũng nhận được cái email **định mệnh**.



Công việc chỉ là developer bán thời gian thôi mà **có test, lại còn phỏng vấn**, sao mà rắc rối dài dòng thế không biết. Do trong mail không nói rõ nội dung test là gì, mình dành ngòi lục và ôn dần đống tài liệu HTML/CSS/Javascript, jQuery, REST, API, thôi thì đủ thür.

Thế rồi, cái ngày **định mệnh** ấy cũng đến. Liệu Hoàng sẽ phải làm bài test như thế nào, trả lời phỏng vấn ra sao? Liệu Hoàng có nhận được công việc “code dạo” mà anh hằng mong ước? Mời các bạn đón đọc trong phần 2 của bài viết =))). Troll đầu năm các bạn đừng chửi nhé.

Chuyện đầu năm – Lần đầu đi phỏng vấn xin việc nơi đất khách quê người (Phần 2)

Posted on 18/02/2016 by Phạm Huy Hoàng

Nội tiếp [phần 1](#), bài viết này sẽ kể về quá trình làm bài test và [phỏng vấn, xin việc](#) code đạo của [chàng coder xấu số](#) tên Hoàng.

Diện đồ nghiêm túc, lên đường

Mồng 3 Tết, ngày 10/2/2015, trong khi bạn bè ở Việt Nam đang vui vẻ ăn Tết, thoái mái vui chơi đập phá ăn nhậu thì mình phải **đi phỏng vấn**. Do tối hôm trước ôn kiến thức tới khuya, lại lo lắng hồi hộp nên đến tận 2h mình mới ngủ được. Cũng may buổi sáng mình dậy sớm nỗi, chẳng muốn ăn nhưng cũng ráng nuốt để buổi trưa khởi đói. Nghe nói bạn bè này formal lắm nên mình cũng không dám mặc style Steve Job áo thun quần jean như thường lệ để đi phỏng vấn. May mà có mang theo 1 bộ vest với đôi giày Tây nên mình cũng có mặc được bộ đồ khá tươm tất. Lịch hẹn là 10h30, mình ra đến tòa nhà ISS lúc 10h25 rồi báo với lễ tân. Chả hiểu các bạn ấy bận hay sao mà tới 10h35 hơn mới có 1 thằng ku xuống dắt mình lên làm test.



Tòa nhà ISS chỗ mình phỏng vấn, trường màu đỏ mà chơi cái tòa nhà đen thuỷ

Ku này cũng khá trẻ nhưng trông hơi dị. Dáng cao, người thon gầy, đeo kính, mặt ngô ngóż, **đúng kiểu nerd/otaku thường gặp** (Lớp mình cũng có một thằng tương tự). Thằng ku dắt mình lên tầng 3, vào 1 căn phòng nhỏ cũng khá xinh, có 2 cái

laptop đặt đối diện nhau. Chẳng lẽ đê test là **solo mid Dota hay Lol** với nó, có 20 phút làm sao đủ?

Hóa ra không phải, bài test cũng khá phù hợp với yêu cầu công việc

Cho một trang web khá đơn giản với html, css, jquery (Sublime Text mở sẵn, muốn dùng IDE gì cũng được). Hãy hiển thị thêm các trường trong dữ liệu, tinh chỉnh giao diện/chọn font cho đẹp mắt, thêm các hiệu ứng animation (UX developer nên yêu cầu phải có khả năng thẩm mĩ).

Đê bài không quá khó, nhưng chắc người ra đê sợ các bạn trẻ hồi hộp làm không kịp nên còn có vài dòng comment gợi ý code chỗ nào, lấy trường gì. Mình đi phỏng vấn đi làm ở VN chưa lần nào bị bắt phải code, thế mà qua đây **xin làm code dạo lại bị bắt viết code** mới vui chứ.

Sau 20 phút làm bài, chưa thấy ai vào nêu mình chém gió với thằng otaku ngồi đối diện. Hóa ra nó là sinh viên năm nhất ngành Computer Science, hiện đang **code back-end** cho dự án, đúng là tuổi trẻ tài cao. Ngoài một hồi thì bé Alice (người gửi thư mời interview cho mình) bước vào phòng và dắt mình vào phòng khác phỏng vấn (Kết quả bài test sẽ được check sau, mình cứ tưởng rót technical là khỏi phỏng vấn chứ).

Bé này *dáng cao, bươi khủng* nhưng vì sợ xâm phạm riêng tư cá nhân nên mình không up hình nhé.

Phỏng vấn gian truân như đánh đố

Bé Alice dắt mình đi lòng vòng gần 3 phút mới tới được phòng phỏng vấn. Team hình như không lớn (Chỉ 8,9 người) mà có tới tận 3 người phỏng vấn mình. **Bé Alice** là Administrator kiêm BA, chuyên lo các vấn đề về account và business. **Bác Brian** (hơi già) là Product Owner, kiêm luôn quản lý group, từng code back-end. **Anh Liam** (khá trẻ) là front-end developer, người chịu trách nhiệm cho mảng front-end của dự án.

Cảm giác đầu tiên của mình là hoi hố. Mình chơi nguyên bộ vest sơ mi quần tây nghiêm túc, còn 2 lão phỏng vấn mình thì... chơi style áo thun quần jean, thiệt là hố không nói nên lời. Mở đầu phỏng vấn cũng khá nhẹ nhàng, không có cảm giác căng thẳng mấy (may mà hồi xưa PV Aswig mình cũng phải dùng tiếng Anh nên giờ không bõ ngỡ). Sau vài câu hỏi giới thiệu bản thân, vì sao em học trường này, ngành có gì hay, sở thích là gì (mình chém từ stackoverflow cho đến blog), các bác bắt đầu hỏi nghiêm túc.



Hôm đó cũng tương tự thế này, nhưng em Alice ngồi bên phải, trẻ đẹp hơn và ngực “khủng” hơn.

Có vẻ là do bài test đã đánh giá được khả năng technical rồi, nên khi phỏng vấn họ không hỏi technical mà tập trung vào **khả năng tư duy + giải quyết vấn đề** hơn. Format khác hoàn toàn với khi mình phỏng vấn ở Việt Nam. Sau đây là một số câu hỏi xoáy đập xoay và những câu trả lời chém gió của mình.

Anh Liam (Front-end developer)

- *Bạn có góp ý gì về bài test? Theo em nó không khó nhưng nên bỏ những phần comment gợi ý, nếu là developer thì sẽ biết viết code ở đâu, như thế nào.*
- *Bạn hoàn thành bài test như thế nào? Dựa theo đè, em xác định những task cần làm, sắp xếp theo độ ưu tiên. Task nào ưu tiên cao thì làm trước, làm dần dần cho đến hết...*
- *Bạn có kinh nghiệm với jQuery hay framework gì ko? Chém về quá trình tự học jQuery, AngularJS, ionicFramework. Thấy các bác gật gù khí thế.*
- *Bạn ấn tượng bởi thiết kế của những trang web nào? Vì sao? Chém trang của Apple vì nó tao nhã, nhiều khoảng trắng, đơn giản và 1 vài trang. Câu này không chuẩn bị, trả lời ko ok lắm.*

Bé Alice (Administrator)

- *Trong quá trình làm việc, làm sao bạn đảm bảo hoàn thành được mục tiêu? Vì dùng Scrum nên mình không cảm đầu code từ đầu tới cuối mà được 4-*

50% sẽ confirm lại với team leader hoặc.. Điều này giúp mình không bị chêch hướng, hoặc hoàn thành sai mục tiêu.

- **Là 1 UX developer, nếu có thể cải thiện UI của Facebook, bạn sẽ làm gì?** Câu này rõ là đánh đố, mình xin 20s xuy nghĩ, ông Product Owner cũng ngồi suy nghĩ. Sau đó mình dành chém là content trên Facebook quá nhiều, đôi khi phải xem những content không muốn xem. Do đó nên có chế độ summary, chỉ hiện cái mình muốn xem. Việc này giúp scroll đỡ mệt. Ông Product Owner cũng gật gù coi như ok.

Bác Brian (Product Owner)

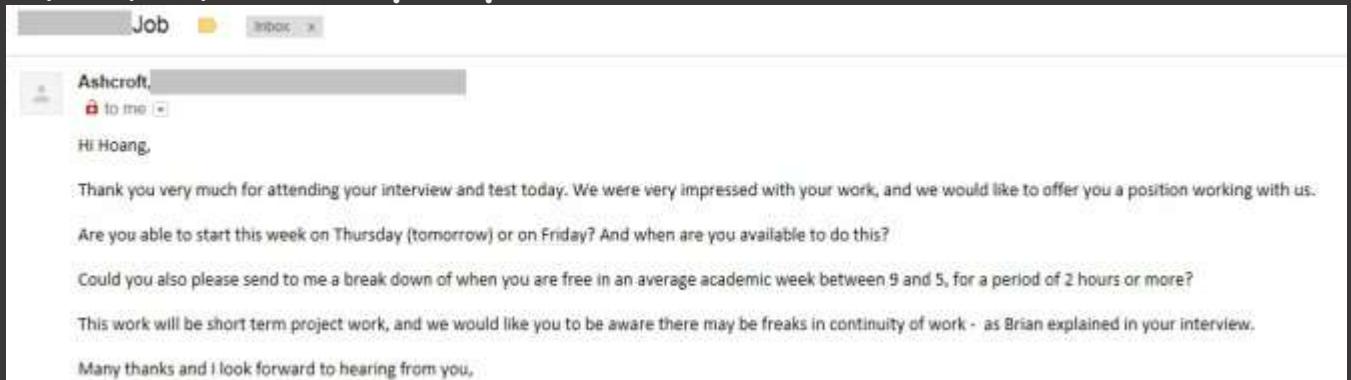
- **Kể về một project mà bạn cảm thấy tự hào?** Lôi cái project Claimbook từ thời làm ở [ASWIG](#) ra chém, mình dựng nguyên đóng UI/ đưa ý tưởng, từ giai đoạn mock-up cho tới demo.
- **Team đang làm ứng dụng iLancaster. Theo bạn cần thêm những chức năng/cải tiến gì?** Câu này thì 2 bên hỏi qua hỏi lại, mình thì chém là phải tìm hiểu người dùng cần gì trước, bên đấy lại đòi nghĩ ra chức năng suông. Chém một hồi thì mình kêu dưới góc độ học sinh, em cần cái A,B,C vì nó sẽ có ích.

Một hồi sau mình cũng được nghe câu “Bạn có câu hỏi gì không?”, mình chỉ hỏi sơ về quy trình làm việc, công việc thường ngày, quy mô team v...v để thể hiện sự nhiệt tình nồng nỗi thôi. Nhớ lời các cụ dạy, sau khi phỏng vấn mình gửi mail cảm ơn, hẹn liên lạc lại sau.



Kết quả

Chắc do thiếu người nên bên này làm ăn khá nhanh gọn “Sau buổi phỏng vấn, chúng tôi sẽ xem kết quả bài test và thông báo kết quả cho bạn trong vòng chiều nay hoặc ngày mai”. Cũng may, sau một hồi mong ngóng thì chiều hôm đó mình cũng nhận được một cái email **định mệnh** khác.



Chắc do ít ứng viên nên quy trình làm việc diễn ra rẹt rẹt. Sáng ngày 10 mình phỏng vấn, chiều ngày 10 có kết quả, gửi mail xác nhận rồi ngay chiều hôm sau xách đít qua ISS đi làm hôm đầu tiên. Hôm đầu đi làm cũng có vài chuyện khá vui nên mình sẽ review ở bài khác nhé.

(Trả lời lun cho các bạn tò mò về lương bỗng, mình làm part time nên **lương tính theo giờ**. 1h được hơn **10 bảng là khoảng 330k**, tuần nào làm nhiều chắc được khoảng 10 tiếng, đủ cafe ăn uống tiêu xài thôi, không đủ trả tiền nhà nữa :'().

Phải làm sao khi viết blog mà ... bí đề tài ???

Posted on 03/02/2016 by Phạm Huy Hoàng

Mình rất vui khi biết một số bạn đã lập blog cá nhân và bắt đầu viết những bài đầu tiên cho blog của mình (Điển hình là thằng Hoàng bạn mình tại: [Codeaholicguy](#)). Một Blog IT cá nhân không chỉ giúp bạn trau dồi [kỹ năng viết, kỹ năng diễn đạt](#) mà còn cũng như tạo điểm nhấn cho [CV](#) khi xin việc.

Khi mới lập blog, hẳn bạn nào cũng tràn đầy [nhiệt huyết đam mê](#), cùng với vô vàn ý tưởng viết hoài không hết. Tuy nhiên, sau một hai tuần, đam mê sẽ tàn lụi dần, các bạn sẽ bắt đầu thấy hơi mệt, bức mình, bí ý tưởng. Đừng lo, không chỉ có các bạn đâu, [hầu hết dân viết lách đều gặp phải tình huống này](#) (kể cả mình). Trong bài viết này, mình sẽ chia sẻ một số cách mình thường làm khi ... bí đề tài viết blog, hi vọng chúng sẽ có ích cho các bạn.



Những điều nên làm khi bí đề tài

- Việc đầu tiên, đó là hãy ngồi vào bàn viết, bật laptop lên, bắt đầu viết một bài với tựa đề “**Phải làm sao khi viết blog mà ... bí đề tài**” như mình đang làm đây, ít ra các bạn sẽ có thêm một bài viết =))).
- Nhớ lại xem gần đây có học được gì mới, đọc được bài viết gì hay hay không. Nếu có hãy đọc/học lại cho kĩ, sau đó tìm tư liệu và bắt đầu viết bài mới. Việc đọc lại, diễn giải lại sẽ làm bạn thấu hiểu vấn đề hơn.

- Muốn viết hay phải chịu khó **đọc nhiều, đọc đủ thứ**. Ngoài sách ra, bạn có thể đọc thêm một số [blog IT](#), hoặc tham gia trả lời câu hỏi trên [stackoverflow](#). Mỗi khi bí, mình thường lang thang webtretho, nhầm, stackoverflow và code project để tìm ý tưởng, săn học kiến thức mới.
- Điều quan trọng nhất là: **phải viết**. Có thể lúc mới ngồi vào máy, bạn sẽ thấy bức bối, khó chịu, ngồi cả nửa tiếng mà tâm trí cứ quanh quẩn đâu đó không viết được. Đừng lo, hãy **kiên trì và ép mình viết**, chỉ cần viết được khoảng 10 phút, bạn sẽ mải mê viết cho hết bài luôn cho xem.
- Nếu vẫn không có ý tưởng và không muốn viết, hãy tạm bỏ qua, bật [IDE lên code](#) hoặc giải trí, làm việc khác khoảng 10-20 phút (Đừng Dota hoặc Lol nhé, mắt cả tiếng đấy >.<). Có mấy tuần mình lo học và mệt không viết được, cuối tuần viết bù 2 bài mệt nghỉ luôn.
- Nếu bí quá, bạn có thể dịch một bài viết nào đó, nhưng mình khuyến khích các bạn chỉ nên lọc một số ý tưởng hay, rồi thêm ý tưởng hoặc kiến giải của mình vào. Bê nguyên xi một bài viết từ tiếng Anh sang tiếng Việt **chỉ giúp bạn tăng khả năng dịch thuật**. Đọc hiểu, phân tích và thêm ý kiến mới giúp bạn dễ “tiêu hóa” những điều tác giả muốn nói.



Những thói quen cần có để duy trì blog

- Mỗi khi có ý tưởng viết bài, hãy **tạo một bài viết mới, viết một vài ý chính**, lưu vào mục draft của wordpress. Khi nào có thời gian rảnh hãy hoàn thành bài viết sau. Có khi cả 2, 3 tuần mình không có ý tưởng nào, có hôm lại có tới 3, 4 ý tưởng trong một ngày. Việc lưu draft lại sẽ khiến bạn không bỏ lỡ ý tưởng hay nào.
- Nên có một lịch đăng bài nhất định, và **có sẵn vài bài viết dự phòng**. Mình luôn dự trữ sẵn 6,7 bài viết để có thể đều đặn đăng bài vào thứ 3, thứ 5. Tối thứ 3, thứ 5 rảnh rồi mình mới viết bài mới. Nhờ có bài dự phòng, bạn không bị áp lực khi viết, lúc không có tâm trạng có thể delay tới 1, 2 hôm sau vẫn được.
- Tạo cho mình một thói quen viết lách, như mình bây giờ đã có thói quen 2 bài/tuần, không bỏ được. Viết nhiều thành quen, ngày xưa mỗi lần mình chỉ viết được khoảng 2-300 chữ là bí, giờ có khi ngồi cả tiếng viết bài gần 7-800 chữ mà vẫn không sao.
- Sau khi viết xong một bài, hãy cho nó vào mục Pending Review, trước khi đăng nhớ rà soát lỗi chính tả, lỗi diễn đạt và chỉnh sửa lại, việc làm này sẽ giúp tăng chất lượng bài viết lên đáng kể đấy ;). Thêm một số hình ảnh vào sẽ làm bài viết bắt mắt, sống động, dễ hiểu hơn.



Nói cho cùng, việc viết blog không phải là một mục tiêu, mà là cả một quá trình, không có kết thúc. Để theo đuổi quá trình này, bạn phải tìm ra niềm vui khi viết. Đối với mình, đó là niềm vui khi hoàn thành một bài viết, đăng một bài viết mới, vui khi có nhiều người bạn xem, comment, hoặc có người đọc blog nêu lên facebook kết bạn với mình.

Nếu chưa có blog, bạn hãy thử ngồi xuống và viết những dòng đầu tiên đi nào, biết đâu bạn sẽ tìm thấy những niềm vui như mình thì sao. Nếu có khó khăn hay thắc mắc gì trong quá trình viết, cứ thoải mái nhắn tin hoặc gửi mail chia sẻ với mình nhé.

Series Javascript sida – Pờ rồ tô tai (Prototype) là cái gì

Posted on 02/02/2016 by Phạm Huy Hoàng

Ở bài trước, mình đã nói về khái niệm object và đít – một số khái niệm cơ bản trong JavaScript. Trong bài này, mình sẽ giải thích khái niệm prototype – một khái niệm khá lồng vòng phức tạp, dễ làm điên đầu các front-end developer.

Prototype là cái đếu gì?

Khi một thằng developer khác cứ đi theo và hỏi bạn “Prototype là cái đếu gì?”, hãy trả lời nó: Là cái đầu **cha** mày, hỏi hỏi suốt. Câu trả lời này có phần hơi bô láo nhưng lại khá là chính xác, có thể hiểu prototype nôm na **là khuôn hoặc là cha** của một object. Trong JavaScript, trừ *null* và *undefined*, toàn bộ các kiểu còn lại đều là object. Các kiểu string, số, boolean lần lượt là object dạng *String*, *Number*, *Boolean*. Mảng là object dạng *Array*, hàm là object dạng *Function*. Prototype của mỗi object chính là cha của nó, cha của String là *String.prototype*, cha của Number là *Number.prototype*, của Array là *Array.prototype*.

Trong JavaScript, **việc kế thừa được hiện thực thông qua prototype**. Khi ta gọi property hoặc function của một object, JavaScript sẽ tìm trong chính object đó, nếu không có thì tìm lên cha của nó. Do đó, ta có thể gọi các hàm *toUpperCase*, *trim* trong String là do các hàm đó đã tồn tại trong *String.prototype*.

```
> String.prototype
< ▼String {length: 0, [[PrimitiveValue]]: ""} ⓘ
▶ anchor: function anchor()
▶ big: function big()
▶ blink: function blink()
▶ bold: function bold()
▶ charAt: function charAt()
▶ charCodeAt: function charCodeAt()
▶ codePointAt: function codePointAt()
▶ concat: function concat()
▶ constructor: function String()
▶ endsWith: function endsWith()
▶ fixed: function fixed()
▶ fontcolor: function fontcolor()
▶ fontsize: function fontsize()
▶ includes: function includes()
▶ indexOf: function indexOf()
▶ italics: function italics()
▶ lastIndexOf: function lastIndexOf()
▶ length: 0
▶ slice: function slice()
▶ split: function split()
▶ substr: function substr()
▶ substring: function substring()
```

Khi ta thêm function cho prototype, toàn bộ những thằng con của nó cũng học được function tương tự.

```
var str = 'abc' // str là string, cha nó là String.prototype  
  
// nhén đói chuỗi đưa vào  
String.prototype duplicate = function() { return this + this; }  
  
console.log(str.duplicate()); // Tìm thấy hàm duplicate trong prototype
```

[view rawprototype1.js](#) hosted with [GitHub](#)

Như mình đã nói, Array, Number hay String có cha là Object, do đó chúng đều có các hàm như *constructor*, *hasOwnProperty*, *toString* thuộc về của *Object.prototype*.

Nhắc lại một chút kiến thức trong bài viết trước về object: Ta có 2 cách để khởi tạo object, đó là sử dụng object literal và Constructor Function. Nếu dùng object literal, object được tạo ra sẽ có prototype là *Object.prototype*. Nếu dùng constructor function, object sẽ có **một prototype mới**, prototype mới này kế thừa *Object.prototype*.

```
var person = {  
  
    firstName: 'Hoang',  
    lastName: 'Pham',  
    showName: function() {  
        console.log(this.firstName + ' ' + this.lastName);  
    }  
}; // object này có prototype là Object.prototype  
  
  
function Person(firstName, lastName) {  
    this.firstName = firstName;  
    this.lastName = lastName;
```

```
this.showName = function() {
    console.log(this.firstName + ' ' + this.lastName);
}

var otherPerson = new Person('Hoang', 'Pham'); // object này có prototype là Person.prototype
// Prototype mới: Person.prototype được tạo ra
// Person.prototype kế thừa Object.prototype
```

[view rawprototype_create.js](#) hosted with [GitHub](#)

Những object được tạo ra bằng cách gọi `new Person()` đều có prototype là `Person.prototype`. Nếu muốn thêm trường hay hàm cho các object này, chỉ cần thêm 1 lần vào prototype là xong. Hiểu nôm na thì prototype cũng **có vài phần giống với class**, mỗi tội sida hơn.

```
function Person(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}

Person.prototype.love = function() { console.log('XXX');

}

var otherPerson = new Person('Hoang', 'Pham'); // object này có prototype là Person.prototype
otherPerson.love(); // XXX
```

[view rawprototype_add.js](#) hosted with [GitHub](#)

Prototype dùng để làm gì?

Tại sao lại đẻ ra cái khái niệm prototype này làm gì? Xin thưa với các bạn, đó là **do sự sida của JavaScript** (Mình đã nói là càng học sẽ càng thấy nó sida mà). Trong

JavaScript **không có khái niệm class**, do vậy, để kế thừa các trường/hàm của một object, ta phải sử dụng prototype.

```
function Person() {  
    this.firstName = 'Per';  
    this.lastName = 'son';  
    this.sayName = function() { return firstName + ' ' + lastName };  
}  
  
// Viết một Constructor Function khác  
  
function SuperMan(firstName, lastName) {  
    this.firstName = firstName;  
    this.lastName = lastName;  
}  
  
// Ta muốn SuperMan sẽ kế thừa các thuộc tính của Person  
// Sử dụng prototype để kế thừa  
SuperMan.prototype = new Person();  
  
// Tạo một object mới bằng Constructor Function  
  
var sm = new SuperMan('Hoang', 'Pham');  
sm.sayName(); // Hoang Pham. Hàm này kế thừa từ prototype của Person
```

[view rawinheriance.js](#) hosted with [GitHub](#)

Nói nôm na, prototype **có phần giống class**, được sử dụng để **hiện thực việc kế thừa**(inheritance) trong JavaScript. Viết tới đây bỗng dung mình chóng mặt hoa mắt rồi, bài hôm nay kết thúc sớm nhé. Ở các bài viết sau, mình sẽ nói về OOP trong JavaScript, rồi các bạn sẽ nhận ra JavaSscript sida đến thế nào.

Bài viết có tham khảo một phần ở: <http://javascriptissexy.com/javascript-prototype-in-plain-detailed-language/>. Một bài viết khác khá hay và hài hước về prototype mà các bạn nên đọc: <http://kipalog.com/posts/prototype-la-khi-gi->

Series JavaScript sida – Object trong JavaScript

Posted on 19/01/2016 by Pham Huy Hoàng

Như đã chia sẻ ở [bài viết trước](#), trong năm 2016 mình sẽ dành thời gian trau dồi kỹ năng front-end, do đó số lượng các bài viết về front-end trên blog sẽ nhiều hơn một chút. Sau series [C# hay ho](#) được nhiều người đón nhận, năm nay blog sẽ có thêm series **Javascript sida**. Lý do là: càng học sẽ càng thấy C# nó hay ho, trong khi đó càng học lại càng thấy Javascript nó sida, bạn nào không tin cứ theo dõi series sẽ biết. Bạn nào theo dõi blog lâu cũng biết mình có một số tình cảm khá phức tạp cả yêu lẫn ghét dành cho [javascript](#). Về bản thân ngôn ngữ, cá nhân mình thấy nó là một ngôn ngữ trời đánh, khá sida, làm bao nhiêu lần mình phải thốt lên *đ.m hay WTF* khi học. Javascript vốn được thiết kế một cách **tạm bợ thô sơ**, dùng để validate ở client side (Bạn nào tò mò muốn biết thêm về lịch sử của js có thể đọc thêm cuốn **Professional JavaScript for Web Developers**). Song chẳng hiểu duyên trời đưa đẩy thế nào, [JavaScript cùng với PHP](#) lại trở thành hai ngôn ngữ được sử dụng ở khắp mọi nơi, dù hùng chúa bao gạch đá.



Tuy ghét thì ghét, đã theo nghiệp [web developer](#) thì trước sau gì cũng phải làm việc hằng ngày với js. Bản thân js tuy xấu, nhưng nó lại đi kèm vô số [thư viện/framework](#) cực kì hay vào hữu ích (jQuery, AngularJS, ...), nhờ NodeJS nó còn lấn sân qua luôn cả back-end. Do đó **học và thuần thực JS** chẳng bao giờ thiệt cả. Nhận ra vốn kiến thức về js của mình còn đôi chỗ lủng củng và thiếu sót, trong series **Javascript sida** mình sẽ viết bài chia sẻ về một số khái niệm từ đơn giản đến

phúc tạp trong js, vừa củng cố kiến thức của mình, vừa giúp ích cho các bạn. Ở bài đầu tiên, mình sẽ giới thiệu kiến thức cơ bản nhất trong JavaScript: object.

Object là khái gì?

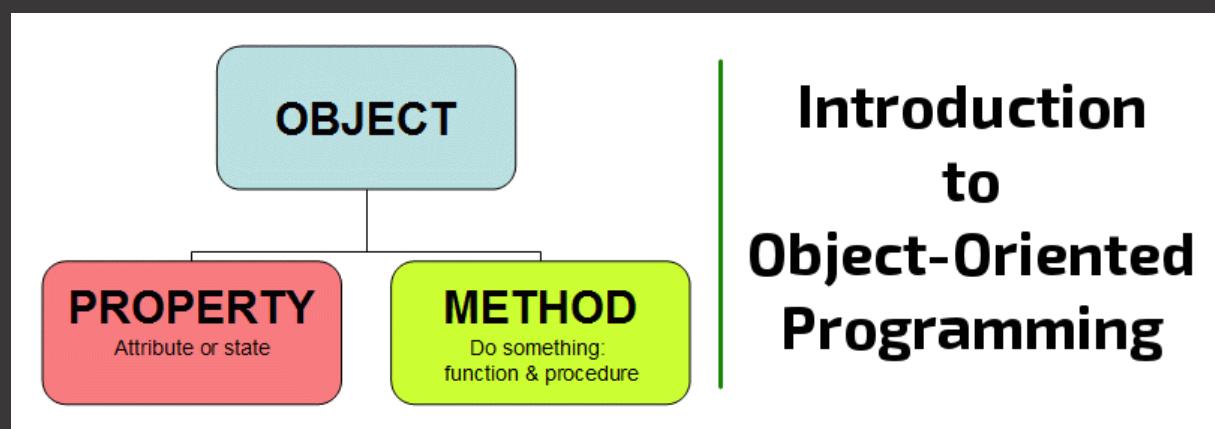
Ai từng học qua môn Lập trình hướng đối tượng đều biết đến các khái niệm Class và Object (Có thể xem lại ở đây: <http://www.qhonline.info/php-nang-cao/57/lap-trinh-huong-doi-tuong-co-ban-ve-nhung-khai-niem.html>). Tuy nhiên, trong JavaScript không có Class mà chỉ có Object, do đó sẽ khiến một số bạn cảm thấy khó hiểu (Đây, mình đã nói bao nhiêu lần là js nó sida rồi mà).

Trong các ngôn ngữ OOP như C++, Java, C#, ... Có thể tạm hiểu Class chính là cái khung, còn Object là vật thể tạo ra dựa vào cái khung đó. Js không có class, ta có thể khởi tạo 1 object mà *không cần xác định class của nó*, có thể hiểu toàn bộ object đều có class chung là *Object*.

Ta có thể hiểu một object là một tập hợp các trường dữ liệu (property) và các hàm (method). Như ví dụ dưới đây, object Student có 2 trường là firstName và lastName, có hàm showName

```
var person = {  
    firstName: 'Hoang',  
    lastName: 'Pham',  
    showName: function () {  
        console.log(this.firstName + ' ' + this.lastName);  
    }  
};
```

[view rawpersonObj.js](#) hosted with by [GitHub](#)



Khởi tạo object thế nào

Trong một số ngôn ngữ khác, để khởi tạo object, ta dùng từ khóa new + tên class. Tuy nhiên, do trong JavaScript không có khái niệm class, ta có thể tạo object theo 1 trong 2 cách sau. Cách khởi tạo object bằng Object Literal thường được sử dụng nhiều hơn.

```
// Cách 1 : Object literal

// Khai báo toàn bộ các trường và hàm

var person = {
    firstName: 'Hoang',
    lastName: 'Pham',
    showName: function() {
        console.log(this.firstName + ' ' + this.lastName);
    }
};

// Cách 2 : Object constructor

var psn = new Object();
psn.firstName = 'Hoang';
psn.lastName = 'Pham';
psn.showName = function() {
    console.log(this.firstName + ' ' + this.lastName);
};
```

[view rawobjCreating.js](http://rawobjCreating.js) hosted with [GitHub](#)

Với các ứng dụng đơn giản, ta có thể tạm dùng 2 cách này. Tuy nhiên, với một số bài toán phức tạp hơn, nếu dùng object literal mỗi lần khởi tạo object sẽ khiến code dài và trùng lặp (Lần nào cũng phải khai báo lại các property và method). Để giải quyết vấn đề này, người ta sử dụng một pattern gọi là Constructor pattern. Một function sẽ đóng vai trò constructor để khởi tạo object (Cách này na ná khai báo class trong các ngôn ngữ khác).

```
function Person(firstName, lastName) {  
    this.firstName = firstName;  
    this.lastName = lastName;  
    this.showName = function() {  
        console.log(this.firstName + ' ' + this.lastName);  
    };  
}  
  
// Khi muốn tạo object person chỉ cần gọi constructor  
  
var psn1 = new Person('Hoang', 'Pham');  
var psn2 = new Person('Hoang', 'Nguyen');
```

[view rawconstructor.js](#) hosted with [GitHub](#)

Một cách khác cũng hay được sử dụng đó là dùng prototype (Mình sẽ nói kĩ hơn về prototype trong những bài sau), nhưng mình thấy đa phần người ta sử dụng Constructor pattern nhiều hơn.

```
function Person() {}  
  
Person.prototype.firstName = 'Hoang';  
Person.prototype.lastName = 'Pham';  
Person.prototype.showName = function() {  
    console.log(this.firstName + ' ' + this.lastName);  
};  
  
// Object được tạo sẽ có sẵn các trường firstName, lastName  
// và hàm showName  
var psn1 = new Person();  
console.log(psn1.firstName); //Hoang
```

```
psn1.showName; //Hoang Pham
```

[view rawprototype.js](#) hosted with [GitHub](#)

Nghịch ngợm với object

1. Truy xuất một trường/hàm của object

Để truy xuất một trường/hàm của object, ta có thể dùng dấu . (dot notation) và dấu [] (bracket notation). Dot notation thường được sử dụng nhiều hơn, nhưng bracket notation có thể làm được nhiều trò hay hơn.

```
var person = {  
    firstName: 'Hoang',  
    lastName: 'Pham',  
    50: 'Hi', // Property có tên là số, không dùng dotNotation được  
    showName: function() {  
        console.log(this.firstName + ' ' + this.lastName);  
    }  
};  
  
console.log(person.firstName); // Hoang  
console.log(person['firstName']); // Hoang  
  
console.log(person[50]); // Bị lỗi  
console.log(person['50']); // Hi  
  
console.log(person.showName()); // Hoang Pham  
console.log(person['showName']()); // Hoang Pham
```

[view rawnotation.js](#) hosted with [GitHub](#)

Để duyệt qua toàn bộ các trường của một object, ta chỉ cần dùng hàm for đơn giản

```
var person = {  
    firstName: 'Hoang',  
    lastName: 'Pham',  
    showName: function() {  
        console.log(this.firstName + ' ' + this.lastName);  
    }  
};  
  
for (var prop in person) {  
    console.log(prop); // firstName, lastName, showName  
}
```

[view rawsampleFor.js](#) hosted with [GitHub](#)

2. Thêm/Xóa một trường/hàm của object

Với các ngôn ngữ static typing như C#, Java, một object được khởi tạo dựa trên class, do đó chúng luôn có các trường và hàm cố định. Tuy nhiên, do JavaScript là ngôn ngữ dynamic typing, ta có thể dễ dàng thêm/xóa các trường trong code

```
var person = {  
    firstName: 'Hoang',  
    lastName: 'Pham',  
    showName: function() {  
        console.log(this.firstName + ' ' + this.lastName);  
    }  
};  
  
delete person.lastName; // Xóa trường lastName  
person.lName = 'Just adding'; // Thêm trường lName
```

```
console.log(person.lastName); // undefined  
console.log(person.lName); // Just adding
```

[view rawaddRemoveField.js](#) hosted with [GitHub](#)

3. Serialize và deserialize

Để giao tiếp với server, JavaScript thường submit dữ liệu dưới dạng pair-value (qua form) hoặc JSON. Do đó, javascript hỗ trợ sẵn việc chuyển object sang chuỗi JSON và ngược lại

```
var person = {  
  
    firstName: 'Hoang',  
  
    lastName: 'Pham',  
  
    showName: function() {  
  
        console.log(this.firstName + ' ' + this.lastName);  
  
    }  
};  
  
// Serialize sẽ làm mất method, chỉ giữ các property  
  
JSON.stringify(person); // '{"firstName":"Hoang","lastName":"Pham"}'  
  
  
  
var jsonString = '{"firstName":"Hoang","lastName":"Pham"}';  
  
var psn = JSON.parse(jsonString); // Chuyển string thành object  
  
console.log(psn.firstName); // Hoang  
  
console.log(psn.lastName); // Pham
```

[view rawserialize.js](#) hosted with [GitHub](#)



Qua bài viết này, các bạn đã có cái nhìn tổng quát về object trong JavaScript. Do sự sida của nó, việc hiện thực các đặc tính của OOP như *inheritance*, *encapsulation*, *polymorphism* trong js khá rắc rối và phức tạp. Các bạn hãy đón xem trong bài viết sau về OOP trong JavaScript nhé.

Bạn nào từng học Java có thể xem thử bài này: <https://codeaholicguy.wordpress.com/2015/12/22/javascript-object-duoi-con-mat-cua-java-developer/>. Bài viết có tham khảo từ nguồn tiếng Anh tại: <http://javascriptissexy.com/javascript-objects-in-detail/>.

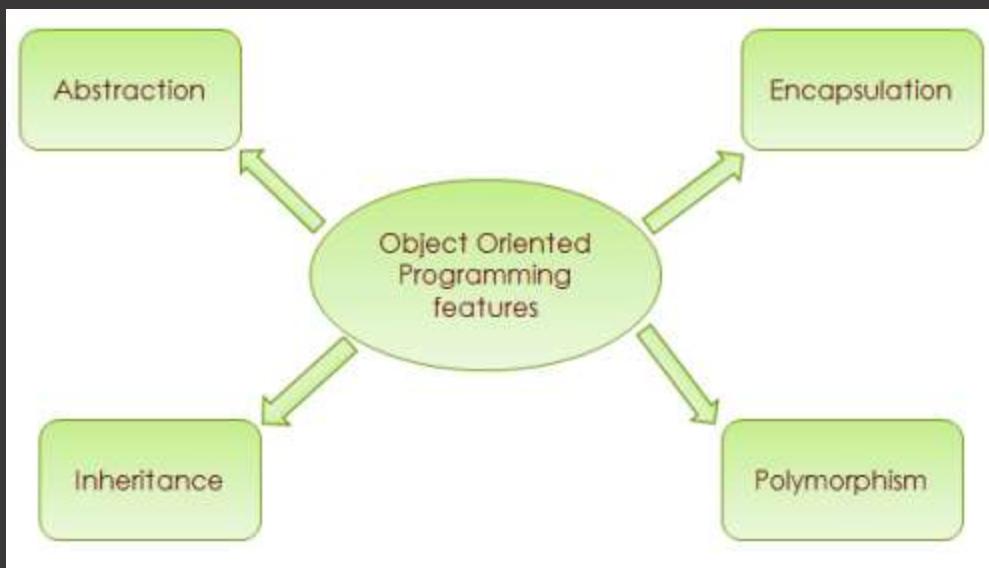
Series Javascript sida – OOP trong JavaScript

Posted on 23/02/2016 by Pham Huy Hoàng

Trước khi xem bài này, các bạn nên ôn lại kiến thức bằng cách xem lại 2 bài viết về object và prototype trong series JavaScript sida. Trước khi phân tích về OOP trong JavaScript, mình sẽ nhắc lại sơ 1 số khái niệm trong OOP. Bạn nào quên rồi có thể lên wiki xem lại nhé: https://vi.wikipedia.org/wiki/Lập_trình_hướng đối_tượng.

Trong phạm vi bài viết, ta sẽ bàn về 3 đặc tính của OOP, so sánh cách hiện thực chúng trong Java và JavaScript. Vì Wiki có sẵn rồi nên mình chỉ copy vào thôi, không giải thích lại lòng vòng nhé:

- **Encapsulation:** Tính bao đóng và che giấu thông tin. Tính chất này không cho phép người sử dụng các đối tượng thay đổi trạng thái nội tại của một đối tượng. Chỉ có các phương thức nội tại của đối tượng cho phép thay đổi trạng thái của nó. Việc cho phép môi trường bên ngoài tác động lên các dữ liệu nội tại của một đối tượng theo cách nào là hoàn toàn tùy thuộc vào người viết mã. Đây là tính chất đảm bảo sự toàn vẹn của đối tượng.
- **Inheritance:** Tính kế thừa. Đặc tính này cho phép một đối tượng có thể có sẵn các đặc tính mà đối tượng khác đã có thông qua kế thừa. Điều này cho phép các đối tượng chia sẻ hay mở rộng các đặc tính sẵn có mà không phải tiến hành định nghĩa lại. Tuy nhiên, không phải ngôn ngữ định hướng đối tượng nào cũng có tính chất này.
- **Polymorphism:** Tính đa hình. Thể hiện thông qua việc gửi các thông điệp (message). Việc gửi các thông điệp này có thể so sánh như việc gọi các hàm bên trong của một đối tượng. Các phương thức dùng trả lời cho một thông điệp sẽ tùy theo đối tượng mà thông điệp đó được gửi tới sẽ có phản ứng khác nhau. Người lập trình có thể định nghĩa một đặc tính (chẳng hạn thông qua tên của các phương thức) cho một loạt các đối tượng gần nhau nhưng khi thi hành thì dùng cùng một tên gọi mà sự thi hành của mỗi đối tượng sẽ tự động xảy ra tương ứng theo đặc tính của từng đối tượng mà không bị nhầm lẫn.



OOP trong Java

Như các bạn đã biết, Java là một ngôn ngữ hướng đối tượng, do đó việc hiện thực các đặc tính OOP rất đơn giản và nhanh gọn, dễ hiểu.

Tính bao đóng trong Java thể hiện bằng cách cho cách khai báo các trường private, chỉ có thể truy xuất trong qua các hàm *get*, *set*.

```

public class Person {

    private String firstName;
    private String lastName;
    private int age;

    public String getFirstName() { return this.firstName; }

    public void setFirstName(String firstName) { this.firstName = firstName; }

    public String getLastname() { return this.lastName; }

    public void setLastName(String lastName) { this.lastName = lastName; }

    public int getAge() { return this.age; }

    public void setAge(int age) { this.age = age; }
}

```

Tính kế thừa và đa hình cũng khá đơn giản, chỉ cần *extend* và viết hàm mới đè lên là xong (các bạn để ý access modifier private và protect nhé, tí mình sẽ so sánh nó với JavaScript).

```
public class Person {  
  
    private int personSecret; // Chỉ person truy xuất được  
    protected int age; // Superman truy xuất được  
  
    public void say(String message)  
    {  
        System.out.println("Person said " + message);  
    }  
}  
  
// Kế thừa và override hàm say  
  
public class Superman extends Person {  
  
    public void say(String message)  
    {  
        System.out.println("Superman said " + message);  
    }  
}
```

[view raw](#) [other.java](#) hosted with

by [GitHub](#)

OOP trong JavaScript

JavaScript thì ngược lại, do **nó vốn sida** nên ta phải áp dụng một số chiêu trò để hiện thực các đặc tính này. Trong JavaScript, để hiện thực tính bao đóng , ta có thể tạo ra 1 *Constructor Function*, đóng gói toàn bộ các trường và hàm vào 1 object. Thông thường, các bạn hay khai báo như sau:

```
function Person(firstName, lastName) {  
  
    this.firstName = firstName;  
    this.lastName = lastName;  
    this.showName = function() {
```

```

        console.log(this.firstName + ' ' + this.lastName);
    );
}

var psn1 = new Person('Hoang', 'Pham');

// các property khai báo vào biến this có thể bị truy xuất từ bên ngoài
// object không còn bao đóng nữa
psn1.firstName = 'changed';
console.log(psn1.firstName);

```

[view rawno_encap.js](#) hosted with [GitHub](#)

Với các khai báo này, tính bao đóng không được đảm bảo. Các property có thể bị truy cập, thay đổi từ bên ngoài. Ở đây, ta phải **sử dụng biến cục bộ**.

```

function Person(firstName, lastName) {
    var fName = firstName;
    var lName = lastName;

    this.setFirstName = function(firstName) { fName = firstName; };
    this.getFirstName = function() { return fName; };

    this.setLastName = function(lastName) { lName = lastName; };
    this.getLastname = function() { return lName; }
}

var psn1 = new Person('Hoang', 'Pham');

console.log(psn1.fstName); // Undefined, không thể truy cập được

```

```
console.log(ps1.getFirstName()); // Hoang
```

[view rawencap.js](#) hosted with [GitHub](#)

Tuy nhiên, biến cục bộ này chỉ có thể truy xuất trong *Constructor Function*, nó tương đương với các trường private trong Java. Trong javascript, không có cách nào để tạo ra các trường protected (Chỉ có thể truy cập từ class kế thừa) như Java và C# được. Việc kế thừa còn sida hon, như mình đã nhắc tới ở bài [prototype](#), trong JavaScript không có từ khóa extends cũng chẳng có class, ta phải sử dụng prototype để kế thừa.

```
function Person() {
    this.firstName = 'Per';
    this.lastName = 'son';
    this.sayName = function() { return firstName + ' ' + lastName };
}

// Viết một Constructor Function khác
function SuperMan(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}

// Ta muốn SuperMan sẽ kế thừa các thuộc tính của Person
// Sử dụng prototype để kế thừa
SuperMan.prototype = new Person();

// Tạo một object mới bằng Constructor Function
var sm = new SuperMan('Hoang', 'Pham');
```

[view rawinheriance.js](#) hosted with [GitHub](#)

Kết luận

Lời khuyên cuối cùng của mình là: Nếu bạn cảm thấy khó chịu với JavaScript, bạn có thể thử học [TypeScript](#), CoffeeScript hoặc ES6. Mình chưa dùbg TypeScript bao giờ nhưng đã xem thử code và syntax thì thấy nó khá rõ ràng, trong sáng và bớt sida hơn JavaScript nhiều (Có class có extends đủ cả).

TypeScript

Walkthrough Inheritance ▾ Share

```
1 class Animal {
2   constructor(name: string) { }
3   move(meters: number) {
4     alert(this.name + " moved " + meters + "m.");
5   }
6 }
7
8 class Snake extends Animal {
9   constructor(name: string) { super(name); }
10  move() {
11    alert("Slithering..."); 
12    super.move();
13  }
14 }
15
16 class Horse extends Animal {
17   constructor(name: string) { super(name); }
18   move() {
19    alert("Galloping..."); 
20    super.move();
21  }
22 }
23
24 var sna = new Snake("Sammy the Python");
25 var tom: Animal = new Horse("Tommy the Palomino");
26
27 sna.move();
28 tom.move();
29
```

1 var __extends = (this || this.__extends) || function (d, b) {
2 for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];
3 d.prototype = b === null ? Object.create(b) : (function (t) {
4 t.prototype = b;
5 return t;
6 })(d);
7 }
8
9 var Animal = (function () {
10 function Animal(name) {
11 this.name = name;
12 }
13 Animal.prototype.move = function (meters) {
14 alert(this.name + " moved " + meters + "m.");
15 };
16 return Animal;
17 })();
18
19 var Snake = (function (_super) {
20 __extends(Snake, _super);
21 function Snake(name) {
22 _super.call(this, name);
23 }
24 Snake.prototype.move = function () {
25 alert("Slithering...");
26 _super.prototype.move.call(this, 5);
27 };
28 return Snake;
29 })();
30
31 var Horse = (function (_super) {
32 __extends(Horse, _super);
33 function Horse(name) {
34 _super.call(this, name);
35 }
36 Horse.prototype.move = function () {
37 alert("Galloping...");
38 _super.prototype.move.call(this, 10);
39 };
40 return Horse;
41 })();
42
43 var sna = new Snake("Sammy the Python");
44 var tom = new Horse("Tommy the Palomino");
45
46 sna.move();
47 tom.move();
48

Các bạn có thể tham khảo thêm ở đây (Máy bài này không có so sánh Java với JavaScript như mình đâu nhé):

Đạo này mình đang hoi bí ý tưởng cho blog, bạn nào có thắc mắc hay yêu cầu gì thì cứ hỏi tự nhiên nhé, nếu thấy yêu cầu hay thì mình sẽ viết bài trả lời. :D

Series JavaScript sida – Bind, Call và Apply trong JavaScript

Posted on 08/03/2016 by Phạm Huy Hoàng

Để hiểu rõ về các hàm được đề cập trong bài viết này, các bạn nên ôn lại chút kiến thức về [object trong javascript](#) và [đít \(this\) trong javascript](#) nhé. Như đã hứa, trong bài này mình sẽ giới thiệu bộ 3 function *bind*, *call*, *apply*. Đây là 3 hàm tạo nên sự mạnh mẽ và bá đạo của JavaScript.

Trói đít (this) lại bằng bind

Bind là một hàm nằm trong *Function.prototype*, do đó chỉ có *function* mới có khả năng gọi nó. Như đã nhắc tới trong bài viết về [this](#), *bind* được dùng để xác định tham số *this* cho một *function*.

The `bind()` method creates a new function that, when called, has its `this` keyword set to the provided value, with a given sequence of arguments preceding any provided when the new function is called.

```
fun.bind(thisArg[, arg1[, arg2[, ...]]])
```

Như trong trường hợp dưới đây, khi ta truyền hàm *showName* vào như một [callback](#) cho hàm *button.click*, giá trị *this* ở đây chính là button đó. Để hàm chạy đúng, ta dùng *bind* để bind giá trị *person* và *this*.

```
var person = {  
    firstName: 'Hoang',  
    lastName: 'Pham',  
    showName: function () {  
        console.log(this.firstName + ' ' + this.lastName);  
    }  
};  
  
//showName truyền vào nhu callback, ở đây this chính là button  
$('#button').click(person.showName);
```

```
// Dùng bind để xác định giá trị this  
$('button').click(person.showName.bind(person)); //this ở đây vẫn là object person
```

[view rawthisSample.js](#) hosted with [GitHub](#)

Không chỉ bind được giá trị this, *bind* còn bind được các tham số truyền vào cho hàm nữa. Do đó, Bind còn được dùng để viết partial function.

Nói một cách đơn giản, partial function tức là tạo ra 1 function mới từ **1 function cũ bằng cách gán mặc định một số tham số cho function cũ đó**. Bạn hãy xem ví dụ cụ thể sau. Mình có một hàm *log* đơn giản có 3 tham số:

```
function log(level, time, message) {  
    console.log(level + '-' + time + ':' + message);  
}
```

[view rawsimpleLog.js](#) hosted with [GitHub](#)

Giả sử mình muốn tạo một hàm *log* khác, ghi lại các log error của hôm nay, mình có thể viết một hàm mới dựa theo hàm *log* cũ:

```
function log(level, time, message) {  
    console.log(level + '-' + time + ':' + message);  
}  
  
function logErrToday(message) {  
    log("Error", "Today", message);  
}  
  
logErrToday("Server die."); // Error - Today: Server die.
```

[view rawlogImproved.js](#) hosted with [GitHub](#)

Thay vì viết như thế, mình có thể viết đơn giản hơn bằng cách dùng *bind*. Ở đây *log* là function cũ, *logErrToday* là function mới, được tạo ra bằng cách gán mặc định 2 tham số *level* và *time*.

```
function log(level, time, message) {  
  console.log(level + '-' + time + ':' + message);  
}  
  
// Không có this nên set this là null  
  
// Set mặc định 2 tham số level và time  
  
var logErrToday = log.bind(null, 'Error', 'Today');  
  
// Hàm này tương ứng với log('Error', 'Today', 'Server die.')  
  
logErrToday("Server die.");  
  
// Error - Today: Server die.
```

[view rawlogBind.js](#) hosted with [GitHub](#)

Partial Function còn được gọi là Curry (Nhiều người bảo 2 cái đó là một, nhiều người bảo 2 cái đấy khác nhau). Nếu bạn thấy khái niệm partial function/curry khá lạ tai cũng đừng lo, chúng ít được dùng trong Java, C# mà hay được sử dụng khá nhiều trong một số ngôn ngữ kiểu lập trình hàm (functional programming) như Haskell, F#, Scala,.... . Functional programming khá là khó học, dễ nhức đầu đau não, bạn nào muôn thử sức thì cứ kiểm ngôn ngữ Haskell mà phang nhé.

Call và Apply, tuy 2 mà 1, thấy 1 mà 2

Đây là 1 cặp anh em... nhầm, chị em song sinh trong JavaScript. Hai hàm này nằm trong prototype của Function (*Function.prototype*), do đó chỉ function mới có thể gọi. Chúng có chung một chức năng lại: Gọi 1 function, xác định tham số this, truyền các tham số còn lại vào.

Điểm khác nhau là *apply* truyền vào một array chứa toàn bộ các tham số còn *call* truyền lần lượt từng tham số. Dễ dẽ nhớ, ta có thể nhầm “A là một Array, C là nhiều Cục“.

The `apply()` method calls a function with a given `this` value and arguments provided as an array (or an [array-like object](#)).

 Note: While the syntax of this function is almost identical to that of `call()`, the fundamental difference is that `call()` accepts an argument list, while `apply()` accepts a single array of arguments.

Syntax

```
fun.apply(thisArg, [argsArray])
```

Parameters

thisArg

The value of `this` provided for the call to `fun`. Note that `this` may not be the actual value seen by the method: if the method is a function in [non-strict mode code](#), `null` and `undefined` will be replaced with the global object, and primitive values will be boxed.

argsArray

An array-like object, specifying the arguments with which `fun` should be called, or `null` or `undefined` if no arguments should be provided to the function. Starting with ECMAScript 5 these arguments can be a generic array-like object instead of an array. See below for [browser compatibility](#) information.

The `call()` method calls a function with a given `this` value and arguments provided individually.

 Note: While the syntax of this function is almost identical to that of `apply()`, the fundamental difference is that `call()` accepts an argument list, while `apply()` accepts a single array of arguments.

Syntax

```
fun.call(thisArg[, arg1[, arg2[, ...]]])
```

Parameters

thisArg

The value of `this` provided for the call to `fun`. Note that `this` may not be the actual value seen by the method: if the method is a function in [non-strict mode code](#), `null` and `undefined` will be replaced with the global object and primitive values will be converted to objects.

arg1, arg2, ...

Arguments for the object.

Cùng xem ví dụ đơn giản này về *call* và *apply*, bạn sẽ hiểu ngay

```
// Tìm max bằng cách gọi hàm Math.max
```

```
Math.max(4, 3, 2, 10);  
  
// Thay vì gọi trực tiếp hàm Math.max, ta có thể dùng call  
  
// Set this bằng null  
  
Math.max.call(null, 4, 3, 2, 10);  
  
// Apply tương tự call, nhưng không truyền lần lượt  
  
// Mà truyền một array chứa toàn bộ các tham số  
  
Math.max.apply(null, [4, 3, 2, 10]);
```

[view rawcallAndApply.js](#) hosted with [GitHub](#)

Call và *apply* thường được dùng để mượn hàm (borrowing function). Các bạn thử đọc xem dòng code dưới đây làm gì?

```
function test(firstParam, secondParam, thirdParam){  
  
    var args = Array.prototype.slice.call(arguments);  
  
    console.log(args);  
  
}  
  
test(1, 2, 3); // [1, 2, 3]
```

[view rawsimpleCall.js](#) hosted with [GitHub](#)

Gợi ý: Arguments là một biến cục bộ trong function, chứa toàn bộ các tham số được truyền vào.

Đáp án: Arguments là một object **giống array nhưng không phải là array** (đây, thấy sida chưa). Arguments giống **array** vì nó có field *length*, có thể truy cập các giá trị nó chứa thông qua index 0,1,2. Tuy nhiên, do arguments **không phải là array** nên nó không thể gọi các hàm của *Array.prototype*.

Do đó, ta phải sử dụng *call/apply* để **mượn một số hàm trong *Array.prototype***, các hàm này sẽ **trả ra một array** cho ta xử lý. **Dòng code phía trên chuyển**

object arguments thành một array. Sao rắc rối quá vậy?? Sao arguments không phải là array luôn đi cho mọi chuyện đơn giản? Lý do là **Javascript nó sida**. Hồi mới biết thẳng arguments này mình cũng muốn chửi thề như vậy đây.

The `arguments` object is not an `Array`. It is similar to an `Array`, but does not have any `Array` properties except `length`. For example, it does not have the `pop` method. However it can be converted to a real `Array`:

```
1 | var args = (arguments.length === 1?[arguments[0]]:Array.apply(null, arguments));
```

Ngoài ra, `call` và `apply` còn được dùng để monkey-patching hoặc tạo spy. Ta có thể mở rộng chức năng của một hàm mà không cần sửa source code của hàm đó. Ví dụ ta có hàm `accessWeb` của object `computer`.

```
var computer = {  
  
    accessWeb : function(site) {  
  
        // Đi tới site nào đó  
  
        console.log ('Go to:' + site);  
  
    }  
};  
  
computer.accessWeb('thiend*a.com'); //Go to: thiend*a.com
```

[view rawmonkeyPatch.js](#) hosted with [GitHub](#)

Sử dụng `call`, ta có thể ghi thêm log trước và sau khi hàm `accessWeb` được gọi mà không can thiệp vào code của hàm đó

```
var computer = {  
  
    accessWeb : function(site) {  
  
        // Đi tới site nào đó  
  
        console.log ('Go to:' + site);  
  
    }  
};
```

```
var oldFunction = computer.accessWeb;

// Tráo function accessWeb bằng hàm mới

computer.accessWeb = function() {
    console.log('Con gà bắt đầu vào web');

    oldFunction.apply(this, arguments); // giữ nguyên hàm cũ

    console.log('Con gà đã vào web');
}

computer.accessWeb('thiend*a.com');

// Con gà bắt đầu vào web
// Go to: thiend*a.com
// Con gà đã vào web
```

[view rawmonkeyPatchCall.js](#) hosted with [GitHub](#)

Nói nhỏ các bạn nghe nè, mấy cái *call/apply* với *bind* này cũng ít người rành, do đó các bạn đọc xong bài này có thể lấy kiến thức ra để lòe thiên hạ. Vì ít người biết nên nếu viết code mà dùng call, apply v...v bạn nhớ giải thích rõ nhé, viết code khó hiểu dễ bị ăn chửi lắm (Nói thật đây, ngày xưa đoạn code *Array.call* gì đó làm mình khó hiểu, đau cả đầu nên phải lôi cả họ nhà thằng code ra mà chửi).

Bài viết có tham khảo một số nguồn ở đây:

- <http://javascriptissexy.com/javascript-apply-call-and-bind-methods-are-essential-for-javascript-professionals/>
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call

- <https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/arguments>
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind
- <http://codeblog.jonskeet.uk/2012/01/30/currying-vs-partial-function-application/>

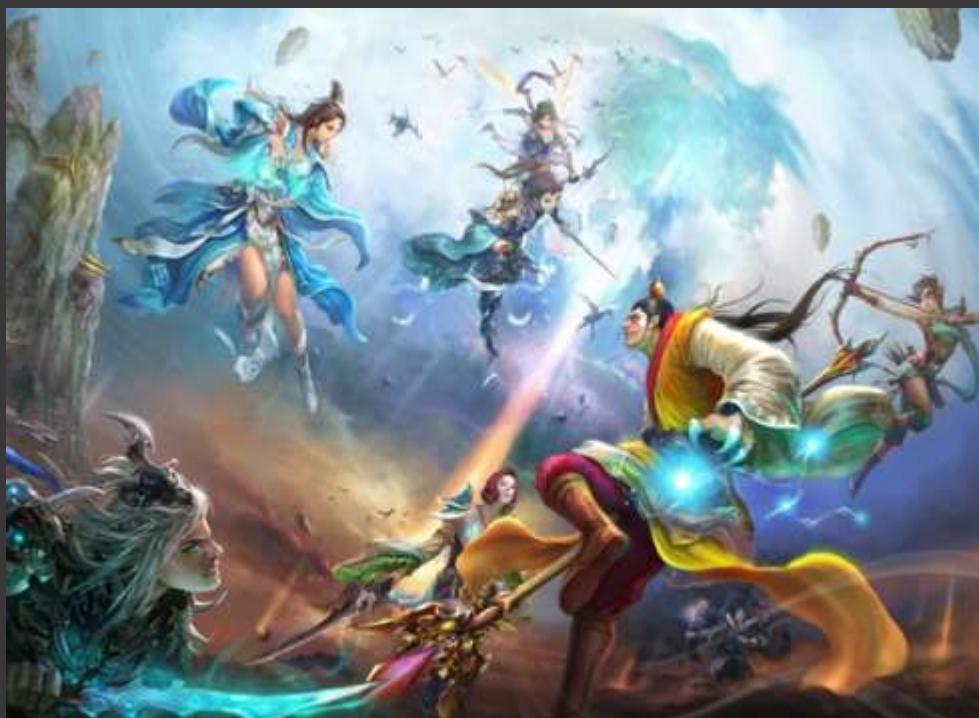
Nhập môn Design Pattern (Phong cách kiêm hiệp)

Posted on 01/03/2016 by Phạm Huy Hoàng

Nhập đề

Kinh thư ghi lại rằng, con đường tu chân có 3 cảnh giới: Luyện khí, Trúc cơ và Kết đan. Luyện khí là quá trình rèn thân luyện thể, cho phàm thân kiên cường dẻo dai. Trúc cơ là quá trình du nhập thiên địa linh khí vào thể nội, giúp khai thông kinh mạch. Khi thiên địa linh khí trong đan điền đạt tới một nồng độ nhất định, sẽ kết thành Kim Đan, đặt bước chân đầu tiên con đường tu chân đại đạo.

Con đường khởi đầu của code học cũng có 3 cảnh giới: Học đồ (Junior Developer), Học sĩ (Developer), Đại sư (Senior Developer). Để đạt đến cảnh giới Đại sư (senior), bất kì Học Sĩ (dev) nào cũng cần phải tường tận vài Design Pattern cơ bản để phòng thân. Bài viết này do tại hạ viết ra trong một phút cao hứng nhất thời, nhằm chia sẻ với các nhân sĩ võ lâm trên con đường truy cầu đại đạo.



Nhiều kẻ khi đạt đến cảnh giới Đại sư (Senior) cứ ngỡ rằng mình đã đạt đến cảnh giới tối cao của võ học mà không biết rằng “Thiên ngoại hữu thiên, nhân ngoại hữu nhân”. Phía trên cảnh giới Đại sư còn có vô số cao thủ đạt tới những cảnh giới khác như Chuồng Môn (Project Manager) hoặc Tông sư (Software Architect). Những kẻ này hiếm thấy như phượng mao lân giác (lông phượng sừng lân), mang một thân võ công cao ngất ngưởng và lương cao ngất ngưởng, lướt gió mà đi, đạp mây mà về. Do cảnh giới bản thân còn thấp, bần đạo tạm thời không bàn tới. Bằng hữu nào hứng thú có thể tìm hiểu thêm [tại đây](#).

Hồi thế gian DS là chi, mà bọn Dev thế nguyên sống chết

Nói một cách đơn giản, design pattern là **các mẫu thiết kế có sẵn, dung để giải quyết một vấn đề**. Áp dụng mẫu thiết kế này sẽ làm code dễ bảo trì, mở rộng hơn (Có thể sẽ khó hiểu hơn 1 chút). Nói văn hoa, design pattern là tinh hoa trong võ học, đã được các bậc tông sư đúc kết, truyền lưu từ đời này qua đời khác. Design pattern là **thiết kế dựa trên code**, nó nằm ở một cảnh giới cao hơn CODE, do đó đệ tử của [bất kì môn phái nào](#)(C#, Java, Python) cũng có thể áp dụng vào được. Ảnh lấy từ bí kíp Head First Design Pattern (xem phía dưới).

A Pattern is a solution to a problem in a context.

That's not the most revealing definition is it? But don't worry, we're going to step through each of these parts, context, problem and solution:

The **context** is the situation in which the pattern applies. This should be a recurring situation.

The **problem** refers to the goal you are trying to achieve in this context, but it also refers to any constraints that occur in the context.

The **solution** is what you're after: a general design that anyone can apply which resolves the goal and set of constraints.

Example: You have a collection of objects.

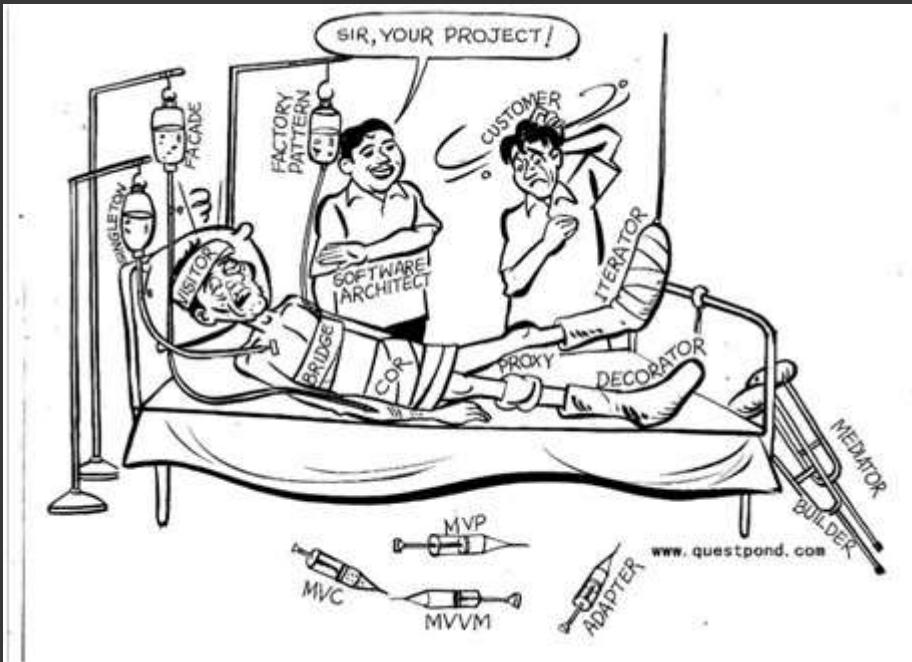
You need to step through the objects without exposing the collection's implementation.

Encapsulate the iteration into a separate class.

This is one of those definitions that takes a while to sink in, but take it one step at a time. Here's a little mnemonic you can repeat to yourself to remember it:

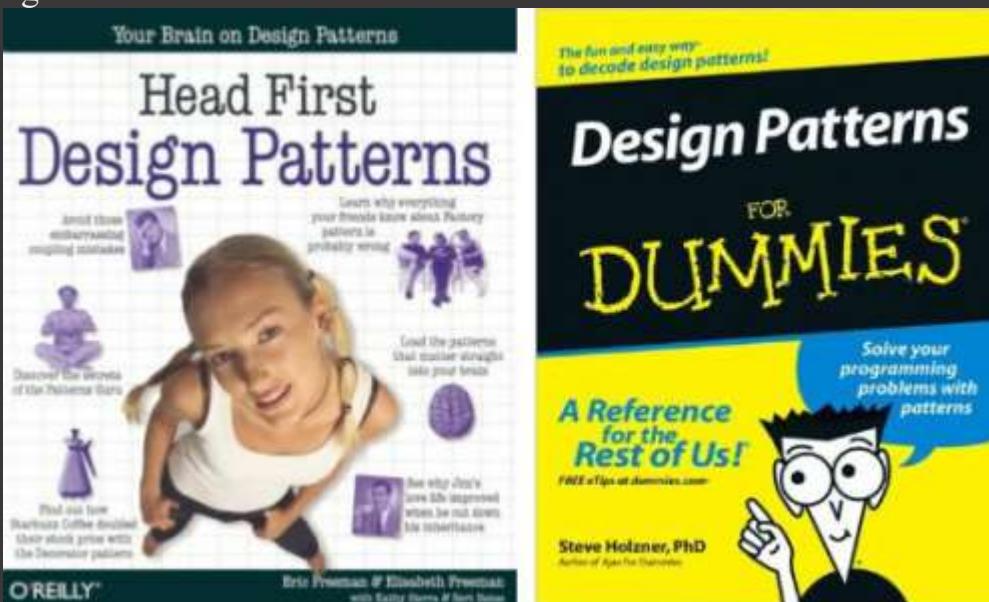
"If you find yourself in a context with a problem that has a goal that is affected by a set of constraints, then you can apply a design that resolves the goal and constraints and leads to a solution."

Trước khi dạy võ, các bậc đạo sư luôn dặn học trò rằng **học võ là để tu thân hành hiệp giúp đời**, không phải để ý vào một thân võ học mà đi bắt nạt kẻ yếu. Nay ta cũng có một lời khuyên tương tự: Học design pattern là để nâng cao trình độ, để **giải quyết vấn đề**, không phải để lấy ra lòe thiên hạ. Nhiều kẻ học nghệ chưa tinh, ngựa non háu đá, nhét design pattern vào dự án một cách vô tội vạ, nhẹ thì tẩu hỏa nhập ma, võ công sụt giảm, nặng thì hồn phi phách tán, vĩnh kiếp không được siêu sinh. Các đao hữu hãy nhìn kẻ than tàn ma bại dưới mà làm gương.



Design Pattern Kiếm Phố

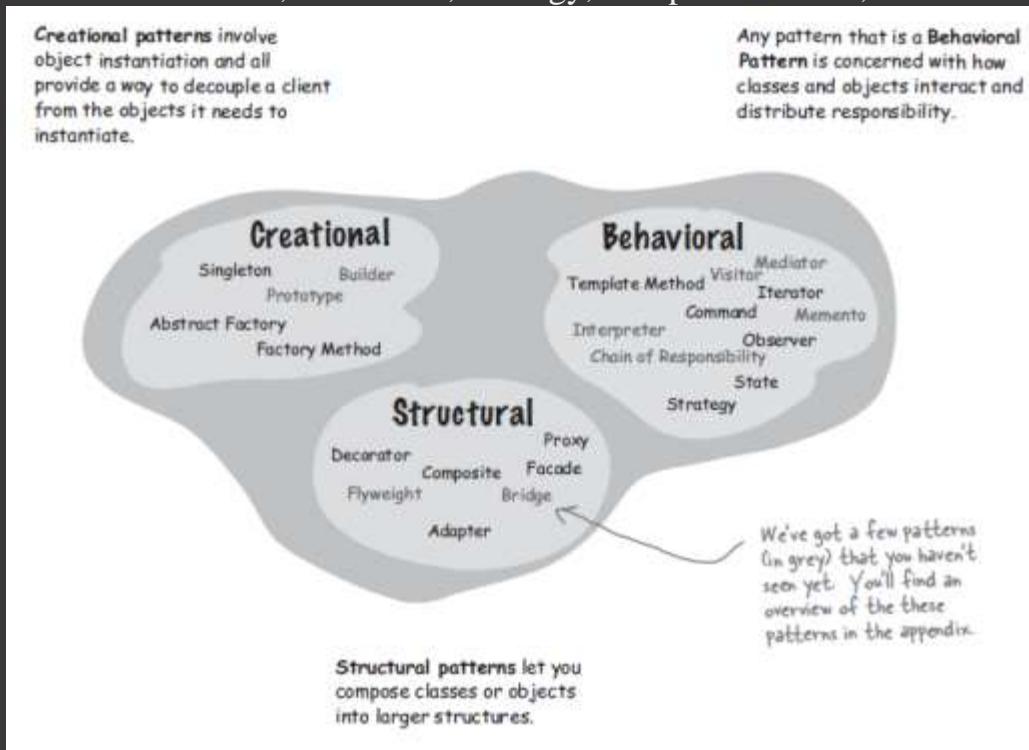
Bí kíp võ công đầu tiên nhắc đến design pattern là [Design Patterns: Elements of Reusable Object-Oriented Software](#). Tuy nhiên, khâu quyết trong quyển này khá khô cứng, khó truyền dạy, do đó các bậc cao nhân đã chỉnh sửa, xuất bản 2 cuốn bí kíp *dễ hiểu hơn* cho hậu thế là **Head First Design Patterns** và **Design Patterns For Dummies**. Thuở xưa khi đặt bước chân đầu tiên trên con đường cầu đạo-cạo đầu, bần đạo cũng tự tu luyện từ hai cuốn bí kiếp này. Các đạo hữu có thể lên mạng tải về ngâm cứu.



Khâu quyết nhập môn Design Pattern

Có khá nhiều chiêu thức design pattern lưu lạc trong chốn giang hồ, song ta có thể tạm phân loại làm Tam Thức:

- **Khởi Thức (Creational Design Pattern):** Liên quan đến việc khởi tạo object. VD: Factory, Object Pool, Abstract Factory, Builder.
- **Cấu Thức (Structure Design Pattern):** Liên quan đến kết cấu, liên hệ giữa các object. VD: Adapter, Bridge, Decorator, Proxy, Composite, Fascade.
- **Vi Thức (Behavioral Design Pattern):** Liên quan tới hành vi của các object. VD: Iterator, Memento, Strategy, Template Method, Visitor.



Khâu quyết một chiêu thức Design Pattern thường có 3 phần. Khi muốn học một design pattern mới, hãy tập trung chú ý vào 3 phần này:

- **Thức Đè:** Vấn đề mà design pattern đó giải quyết
- **Thức Đồ:** Sơ đồ UML mô tả design pattern
- **Thức Phổ:** Code minh họa

Dưới đây là một Design pattern đơn giản nhất mà hầu như học sĩ nào cũng biết: Đơn Thân Độc Mã, thuộc Khởi Thúc, hay còn gọi là Singleton, thuộc loại Creational Design Pattern.

- **Thức Đè:** Design Pattern này được dùng khi ta muốn đảm bảo **chỉ có duy nhất một object được sinh ra trong toàn hệ thống**.

Singleton	
-	instance : Singleton = null
+	getInstance() : Singleton
-	Singleton() : void

- **Thức Đồ:**
- **Thức Phô:**

```
public class Singleton {  
    private static final Singleton INSTANCE = new Singleton();  
  
    private Singleton() {}  
  
    public static Singleton getInstance() {  
        return INSTANCE;  
    }  
}
```

[view rawsingleton.java](#) hosted with by [GitHub](#)

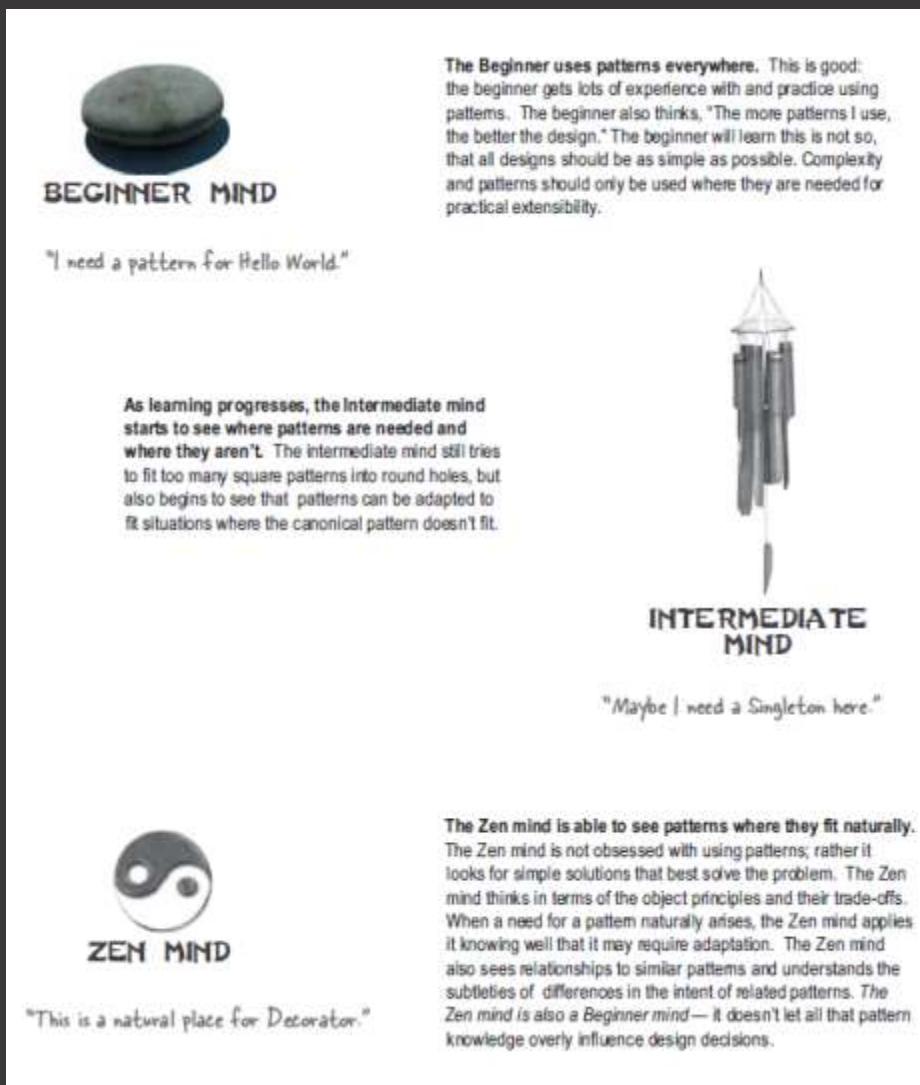
Thay lời kết

Xin nhắc lại một lần nữa: *Design Pattern được tạo ra để giải quyết vấn đề, chứ không phải để phức tạp hóa nó*. Các bậc cao nhân có câu: nước có thể dâng thuyền, cũng có thể lật thuyền. Design Pattern có thể giải quyết vấn đề, cũng có thể làm nó rắc rối phức tạp hơn.

Kẻ sĩ dùng design pattern cũng chia làm ba cảnh giới. Kẻ sơ nhập thì nhìn đâu cũng thấy pattern, chỉ lo áp dụng, nhét rất nhiều pattern vào mà không quan tâm đến thiết kế. Lăn lộn giang hồ một thời gian, đến cảnh giới cao thủ, sẽ học được rằng khi nào

cần dùng pattern, khi nào không. Đến cấp bậc đại sư, chỉ dùng pattern khi đã rõ lợi hại của nó, biết lấy sự đơn giản hài hòa của design tổng thể làm trọng. Có thể tổng kết quá trình này bằng một câu:

Khi chưa học đạo, ta thấy núi là núi, sông là sông. Khi mới học đạo, ta thấy núi không phải là núi, sông không phải là sông. Sau khi học đạo, ta lại thấy núi chỉ là núi, sông chỉ là sông.



Đây là bài viết đầu tiên của blog trong series về design pattern. Do là bài đầu nên viết theo phong cách kiêm hiệp để thu hút bạn đọc thôi, những bài viết sau trong series sẽ quay lại phong cách bình thường nhé. Chém gió tốn nhiều no ron não lắm, cứ viết

kiểu này có khi tác giả tâu hỏa nhập ma, hồn phi phách tán chứ chẳng chơi :'. Bạn nào thích bài viết theo phong cách này có thể đọc bài [Luân về comment code](#) nhé. Ngoài ebook, các bạn có thể tìm hiểu thêm về design pattern ở đây.

- https://en.wikipedia.org/wiki/Software_design_pattern
- https://sourcemaking.com/design_patterns