The NeceMoon Album

Technologies and Strategies to Keep Moving Forward

Chapter 6
Software Development and Engineering

Featuring Darren Mart
and Jean-Patrick Ehouman

By Necemon Yai

First Edition TheAlbum.NeceMoon.com Evasium ®

The NeceMoon Album

Technologies and Strategies to Keep Moving Forward

By Necemon Yai

First edition

Published by Evasium ®

April 2018

London, UK

TheAlbum.NeceMoon.com

© Necemon Yai

necemon@gmail.com

www.necemonyai.com

Version 1.0.7.235

To the ones I lost, to the ones I got back

For each one that gets lost, ten get laid back

Table of Contents

Introduction

About The NeceMoon Album: what is this all about?

The NeceMoon™ is a Blog about Technology and Strategy. The Album is The Best-Of, a compilation of the most popular articles on The NeceMoon™.

The main objective of The NeceMoon™ is to share tips and insights on a sensible range of topics, in order to let others learn from my mistakes and successes, and hopefully to make things easier for the next person. The main objective of The Album is to promote an optimal access to that content. The Blog format does not always do justice to techno-strategic content. Originally, Blogs were designed in a journalistic spirit and are more suitable to chronological events and (more or less trivial) discussions around daily news. Even if the usefulness and the importance of an analysis persist in time, it becomes almost impossible to find and difficult to consult, as more articles keep stacking.
That is why the best articles have been cherry-picked, reviewed and arranged in a logical order that better matches the layout of a book. The Album is free of charge.

The NeceMoon™ can be accessed from NeceMoon.com (or necemonyai.com/blog)
The NeceMoon™ Album can be downloaded in various file formats, in full from TheAlbum.NeceMoon.com (or necemonyai.com/Blog/page/The-Album.aspx). The available formats are PDF, EPUB, MOBI/AZW3/KF8 (Amazon Kindle) and MP3. Furthermore, the various chapters and volumes can be downloaded independently/separately, according to your interests.

About the Author: who is Necemon Yai?

I am a Software Development Engineer extensively involved in Microsoft .NET technologies. Full time developer. Part time digital artist, strategist, essayist and entreprenerd. I majored in computer science from NIIT, Christ University and Swansea University (Master of Engineering, Computing).
At the time of publishing this, I have worked for a Europe leading E-Commerce company, a major UK financial group, the General Electric global corporation and a few tech start-ups that you probably never heard of.
Over the past decade or so, I have been running The NeceMoon blog, where I describe my experiences, my observations and my reflexions. I mostly talk about Technology and Strategy. Here I share my most popular articles.

About the Contributors: who is in your War Council?

I invited the best writers in my network to include some contributions in this book, especially some of their most relevant insights in terms of Technology and Strategy. These top authors are, Ahou The African Chick, Antoine Mian, Cyriac Gbogou, Darren Mart, Edith Brou, Holty Sow, Israel Yoroba, Jean Luc Houedanou, Jean-Patrick Ehouman, Karen Kakou, Monty Oum, Nanda Seye, Nnenna Nwakanma, Olivier Madiba, Vanessa Lecosson and Yehni Djidji.
Along with their respective writings, you can find links to their own web pages. In addition, most of these contributors introduce themselves and provide you with some tactics in our exclusive interviews that you will also find in this book.

About You, Dear Reader: who is this book for? What's in for you?

In The Album, there are 9 chapters organised in 2 volumes. Each chapter deals with a specific topic. You don't have to read everything. If you are interested (to one extend or another) in one or more of these topics, you would possibly appreciate the related chapter(s):

Volume 1: Moon Light (softcore)
- Chapter 1: Strategy and Tactics
- Chapter 2: Digital Marketing and Web Visualisation
- Chapter 3: Corporate Worlds and Emerging Markets
- Chapter 4: Quick Wins, Tricks and Tips
- Chapter 5: Transition - Extra Thoughts and Sharp Fantasy

Volume 2: Full Moon (hardcore)
- Chapter 6: Software Development and Engineering
- Chapter 7: C# .NET Programming
- Chapter 8: Epic Prototypes, Classic Projects, Historic Genre
- Chapter 9: Research and Case Studies

If you want to, you can download and read only the chapter(s) and volume(s) that you are interested in. Several file formats are available on TheAlbum.NeceMoon.com
(or necemonyai.com/blog/page/The-Album.aspx)
All the Web links in this document are working, feel very welcome to click on them.

For All Practical Purposes

This document contains the Chapter 6 of The Album: “Software Development and Engineering”. If you care, 8 other chapters are also available. Depending on your interests, you may download, (re-)read or share any of the various chapters and volumes independently/separately. The PDF, EPUB, MOBI/AZW3/KF8 (Amazon Kindle) and MP3 formats are available.

To get them, just click on any of the links you like below or go to TheAlbum.NeceMoon.com
(or necemonyai.com/Blog/page/The-Album.aspx)

The Album is available in French as well at Album.NeceMoon.com
(or necemonyai.com/Blog/page/L-Album.aspx)

Chapter 6

Software Development and Engineering

Featuring Darren Mart and Jean-Patrick Ehouman

8 reasons why you would enjoy being a programmer

By Necemon

If you are a programmer, there are probably a lot of reasons that motivate you to do what you do. I hope you find here some more motivations.

If you want to be a programmer, you may find here some more reasons to go for that awesome path.

However, I guess I am writing this mainly for those who don't know what to do with their life. Here are some clues about a job that is fun, useful and satisfying to many extents.

From the top of my mind, here are 8 reasons why you would enjoy being a programmer. I hope this inspire you.

1. Programming makes your dreams come true. When you understand programming, you can give life to your thoughts by applying them to real life. You can literally create things.

2. Programming is the ultimate form of interactive art. You can make software, websites and games that others can play with. So you can talk to them indirectly and they can talk back. No other art form is quite this interactive. While drawing, painting, movies and music go to the audience (in one direction), code goes both ways.

3. That's the kind of job you can do from anywhere. From your couch, from home, from office, on travel, whichever country... The only things you need are a computer and your brain.

4. It's easy to learn. There are tons of ressources available online, many of them are free. Also, there are many online communities that can support you through forums, chats, emails, etc.

5. You don't have to rely on anyone to do programming. You may have a chance to do it for a company or for a research programme. Even if it's not in a corporate job, you can work in a team. And even if you don't find a team that fits you, you can work by yourself. Also, you don't need much money to get started.

6. You can't have enough of it. You can't get bored. Requirements and technology are moving up so fast that you always face new challenges.

7. It's a field of meritocracy. They know you don't fake your skills. You know what you know. You do what you can do and you get a fair recognition for it.

8. It's fun !

N.

Writing and Programming: Pretty Much The Same Thing

By Jean-Patrick Ehouman

5 years ago, if I was told that I could run a blog, I would not have believed it. As a software engineer, I was spending more time designing and writing programs. However, a priori, nothing suggested that I could write good articles too.

In hindsight, I can say that these two activities have more similarities than differentiation points. When you write, you create, you fill a blank, you give life. Similarly, when you design a program, you create a system that will be used on a daily basis. So in both cases you need to deeply understand the reader or the user of your creation.

Putting yourself in their shoes leads you to imaginative and creative like a painter or a pianist. So, when I am asked about the fundamentals required to learn how to make good computer programs or software, I ask the interested party if he has ever written a letter, a short story, an article, etc. If the answer is "yes", then the person already has prerequisites to learn how to write programs. If not, they can still try to write a short story and assess their abilities to create or innovate.

Coding is just writing.

On Technical Orientation : 5 Basic Considerations When Starting To Code

By Necemon

A fresher sent me the following query:
Hi Necemon, thank you for accepting my invite. I am new to the IT field and I would appreciate if you could give me some guidance. Based on your work experience, can you tell me what companies look for in a computer guy ?
*I heard about you from senior students at **Christ University**, I am in Bangalore and I like computer science but I do not know what to learn and how to begin.*
*Actually, I think **I like programming** but I am told that the C language is no longer relevant, and **I am also told about Ruby, C#, Python, etc**. I'm confused.*

Please don't get confused. Technology is simply a way to solve a problem or to achieve a goal. What is your goal?
Create apps? What apps do you want to create and why?
It's a bit as if you come to me to ask me what vehicle you should be driving. If I ask you what you want to do with that vehicle, you wouldn't just tell me that you just want to move away, right? I know you want to move... My question is, where are you going?

What companies are looking for ? Ok, I fully understand what you are asking here. You want to make sure your education will guaranty an interesting job later in the IT development Industry. Obviously, I could tell you that a certain technology T is in high demand right now, but it's not that simple. There are a few other things to consider:

1. The requirements may vary with location (country or region). The hottest jobs in the US are not necessarily that popular in India. So unless you know already where you are going to work, it's not that easy to target on a trend basis.

2. The demand changes with time. What is relevant today may not be (as) prevalent tomorrow. The technologies evolve and replace each other. So what's fashionable now might be different from what will be popular by the time you get your degree.

3. You might not like the technology in vogue or the uses of that technology. If I tell you that a given technology T is in high demand, that it allows you to locate and fix bugs / errors in a super boring / huge / complicated banking system, plus there are plenty of calculations... What if you don't like calculations? Are you still going to embrace this technology and to accept this path for the rest of your career?

4. Even if we consider only one city and a given time, various companies are seeking different things, depending on what they do. There is no perfect technology that is better than all others in all areas. Each technology has its own sets of advantages and disadvantages. C and C ++ may be better than Ruby at a few things (and vice versa), Python is better than C# in some respect (and vice versa), etc.

5. As I said above, the technology is just a mean to get somewhere. When you consider Facebook for example, most users don't care if it was built with PHP, C, Java, Perl, C # or Python. What is important for people is, how the site or application can help them in their lives.

I think that's where you should start. What strengths and assets do you already have? (don't tell me you don't have any). What contribution do you intend to come up with for your family, your friends, your community, your country, and for the world? And what do you expect in return?

If you do not know what to do with your life, some time ago I wrote an article that might inspire you : What will you do in life? Take time to reflect on your ambitions and we can talk about the resources you will need.

If you know WHAT you want to do, it would be easier for me to tell you HOW to do it.

Let's speak soon,

N.

UPDATE - Shabbir Kahodawala shared a few clever insights on this matter :

I agree very much with your response to the fresher - about the need to realise and work on his talents and concentrate on writing good code and immersive UI.
I would like to add a few points as well taking the perspective that every Indian student goes through the same dilemma due to lack of job oriented education, because institutions focus on technical oriented education.

IT is not only about writing code. Think of it as a factory where there is Marketing, Client Requirement gathering, Planning, Product development, Product Testing, Infrastructure planning, Product Deployment, Product Maintenance, Customer Support, Issue Resolution and Product Improvements.

Each of these creates many job opportunities for IT students and one needs to understand where his strengths lie. He can do that by talking to IT professionals who can introduce to other professionals in each department and willing to share what skills they require on today's world. That will make his goals more clear.

Secondly, a competitive IT professional should always have his basics right. To be able to write good and neat code. He should be always able to visualize a requirement into an algorithm and then the language syntax that he uses can always change. That's why colleges teach C & Java (object oriented) because these help a coder develop his basics about Data Handling, Functions, Objects, Classes and runtime environment.

4 Substantial Tactics Guaranteed To Boost Your Game Creation Skills

By Necemon

A user of one of my web applications sent in the following question:
Hi my name is T. and I am 15, when I am older I want to be a part of the gaming industry and I am just wondering can you guys help with that, can you give me any tips or any experience?
Would posting to your website help me out in this? I would love to hear your reply, it would really help me out.

Here was my reply:

There is a lot to say on the topic but for today I will try to keep it simple and give you some practical advice you can start using right away.

1. **Read a lot on the topics you are interested in.** Read every day if you can. There are tons of free resources available online, some of my favorite sites and blogs for game creation are:
gdne.ws
lostgarden.com
procworld.blogspot.co.uk
whatgamesare.com
gamestudies.org
designer-notes.com
higherorderfun.com
gamecareerguide.com
webwargaming.org
raphkoster.com/gaming
gamedevelopment.tutsplus.com
nwn.blogs.com

2. **Establish specific goals.** You may have noticed that in point 1, I only gave you a small portion of what's out there, but it's quite an extensive read already. We are living exponential times, you can't do and learn everything by yourself at once. You have to be specific about what you want. Vague plans provoke vague results. What is it that you want to do in the video game industry? There are so many options, you may want to do some research on your options.

3. **Pick your stars.** To bounce back from the previous point, after you make a list of the skills you want to master, do some research on people who are already really, really good at that. Find some role models, see how they started and how they made it. Combine their tricks and throw in your own style to develop your own skills. It may take months but if you practice consistently, you will get really good too. Some of the people you can check out:

Game development:
- Shigeru Miyamoto
- Hideo Kojima
- Notch (Markus Persson)

Digital artists:
- Akira Toriyama
- Masashi Kishimoto
- Monty Oum

Scenario writers:
- J.K. Rowling
- Stephen King
- Tom Clancy

4. **Make a game.** No, you are not too young to start. It doesn't have to be anything big and right now, you can at least start learning. The earlier you start, the more time you will have to practice and the sooner you will become excellent. You can start with something very basic or at least start reading about it. If you want advice on how to start, how to find tutorials and resources, I can help.
On a related note, I made a game myself : babifraya.com It's not great and I am working on an improved version. But the important thing is to start and keep practicing :-)

In short, educate yourself consistently.

As to whether posting to degammage.com would help, my honest answer is: maybe.
You see, it's a new project with a small community and it's too early to say how successful it will be. However, there will be some fresh content everyday, so there will be things for you to learn, and you may also visit flux.evasium.com which is even more into virtual worlds, education and technology.
Your Evasium account will work on there too.

Additionally, you may want to join other game forums and communities, like the ones I listed above. The more you learn, the better!

10 Years of Programming

By Necemon

I started programming when I was in high school. I was taught Pascal and HTML among other things. But I was also taking some summer programming classes in an IT institute. That's where I learned Visual Basic and software/database design (during those days, we were using MERISE (Méthode d'Etude et de Réalisation Informatique pour les Systèmes d'Entreprise)). I eventually got Visual Basic installed on my home computer and a book from which I could practice alone, with books.
But, it's only after I graduated from high school, that I started learning and applying C# almost full time during higher studies, and that's what I considered to be my real start into programming and software engineering.
I learned a few lessons over the past decade, and I thought I would take a moment to gather my thoughts on these things. It took me about ten years and a lot of experimentation to figure out some of this.

1. Learning a programming language is the easy part: be aware and beware of the platforms
Take C# for example. Learning the C# language is not difficult. If you already have a good understanding of computer language fundamentals, and if you have some experience in other object-oriented languages, you can become a competent C# programmer within a few days, at least as far as the language itself is concerned. However, the real price to pay is not about the language, it's about the platform. To develop with C# on .NET, you need to know:

- the .NET framework
- one or more .NET technologies such as ASP.NET or WPF
- and the Visual Studio development environment.

The time required to become proficient in .NET development is usually measured in months, even for an experienced developer. Learning a platform is always more expensive than learning a specific language, therefore choosing the platform is the most crucial decision.
Learning always has a cost and this cost is one of the key factors to consider when choosing the technology you want to learn. The real cost of learning is in time, learning always takes time. Since you do not have time to learn everything, it is important to think strategically about what you want to learn. And since the languages are easy, it's the platforms that you have to be careful about: the technologies associated with the language, the development and deployment tools, the operating systems, and other infrastructures.

2. I repeat, learning a programming language is the easy part: meet the underlying concepts of software engineering

The syntax itself, the words you use when applying the language are relatively simple and you can easily pick them up as you go. However, that's far from being enough for producing quality code, which often involve OOP and SOLID principles, unit testing, design patterns, TDD, BDD, and other technical concepts which are beyond the scope of this article. Anyway...

3. Actually writing code is only one (small) part of the job

A software engineer is often expected to be involved into technology research, tools and projects configurations, DevOps and admin tasks, debugging and testing procedures, documentation, and technical debt (fixing and refactoring existing code). Also, they have to be thinking about solutions and designing systems : sometimes the most important work is done away from a keyboard.

4. Tried and true : old and boring is sometimes the best.

It's not so much old vs new, or cool vs boring, but rather the thing you are most experienced with. As they say, I trust not the developer who practiced 1000 technologies once, but I trust the developer who practiced the relevant technology 1000 times. If the goal is to "just build the damn thing", go with a stack you would be most productive in.

For example, one of my contacts is making $25,000 per month with a SaaS that was built on boring ASP.NET+SQL Server+Angular 1 because that's what he knew. he hosts it on Windows because he knows how to make it fast and secure. He succeeded by focusing all his time on building the features that clients were asking for, instead of learning fancy tech.

It's important to realise that the technological treadmill never stops. Yet another JavaScript framework could have been launched while you are reading this. Today's cutting edge tech didn't even exist when I was getting started (EF Code First, Xamarin, ASP.NET Core, Razor), and this leads us to the 2 next points.

5. Focus on sustainable technologies

The only constant in the world is change. Actions and time management is an important skill for developers, particularly because we are on a technological treadmill that keeps moving or even accelerating.

For example, Web technologies that were popular around the year 2000 (Flash, ASP Classic and Java Applets) are becoming almost obsolete and decreasingly marketable. Today, we are talking about ASP.NET Core, SignalR, Angular2, React and VueJS. None of its technologies existed in the year 2000, and these new technologies are likely to be obsolete within 10 years.

What hasn't really changed? The fundamentals of languages such as C++/C#, their implementations of algorithms and their principles are still relevant after several decades. If you master the basics of a stable system, you could adapt to change better, you could appreciate it and use it to evolve.

6. Balance between exploration and exploitation

Exploration is about learning new things, studying new techniques, reading books, watching video tutorials, practicing and improving skills. Exploitation however, is to take advantage of what we already know to solve real life issues. It's about thinking creatively about ways to use the knowledge we already have to create value for others.

So yes, both of these tasks are both necessary and important. The risk is to be too focused on either activity.

Too much exploration, and you will never achieve a useful level of expertise in the chosen technology. There is a huge opportunity cost with this kind of light learning, because, although it expands your mind, the time it takes implies that you don't really improve on the skills that you have already acquired.

On the other hand, too much exploitation can keep you from evolving in new technologies, and can limit your employment opportunities.

7. It’s easy to be great... It’s hard to be consistent

It's easy to be great for 2 minutes. It's hard to stay great constantly, every day.

When you have a good idea for a new project, you feel a great desire to start researching, designing and programming. You feel a rush to turn your idea into something real and you become super productive. But the problem is that this motivation fades over time.

Yes, it's fun and it's easy to have new ideas and start working on them. But then there are the efforts to be made, the adjustments, the launch, the maintenance, the corrections, the improvements, and so on. Over several months. This is where it gets hard. It is hard to stay focused on the same idea, on the same project for months and years. It takes a lot of discipline.

It’s easy to be great. It’s hard to be consistent.

8. Diversify your skills

Don't be just a programmer, become an Expert Who Programs, an expert in another relevant field that you are passionate about. You can be an entrepreneur, a project manager, a Big Data scientist, a researcher, a security specialist, etc. If you are an Expert Who Program, in addition to being able to program (maybe full-time), you also have an additional credibility that is related to something other than software engineering.

Hence the importance of getting an education. If you go to university and you already know how to program, you probably won't learn much about programming. That does not mean you should not go to these schools. You will need some culture, and universities are great places to get that. You acquire culture by studying and understanding the world that humans have created, from different angles. It would be difficult to acquire this kind of knowledge if you do nothing but study programming.

9. Pick your niches and standout

The smaller the niche you choose, the greater your chance of being viewed as a standout in your field. It's very hard for a developer to become a standout in "PHP Web development". They're great, versatile, useful, but not noteworthy. One developer who knows how to work with these technologies, feels they are easily replaceable because there are so many out there with a comparable skill set. These areas are too broad for you to easily standout from the pack. If, on the other hand, you become known for a niche, like Xamarin.Forms or JavaScript Visualisations you're much more likely to be valuable to those looking specifically for that skills set.

10. Age of Skills

Information is the specific knowledge that you need to solve problems. Skills represent the ability to implement solutions using your knowledge.

In a world where most of the knowledge and tools are virtually free, what makes the difference? The skills, of course. We are no longer a knowledge-based society, we are a skills-based society. There was a time when almost all university degrees guaranteed a good job. Now this is no longer the case. Nobody cares about what you know. People care about what you can do. They pay you to do things, not to know things.

Teach yourself programming in 10 years.

Researchers have shown it takes about ten years (or 10 000 hours) to develop expertise in a field.

The solution is reflective practice: it's not enough to repeat the same things over and over again, but challenging yourself with a task that exceeds your current ability, to try it, to analyze its performances during and after, and to correct all error. Then repeat. And repeat again. It seems there are no real shortcuts. Learning through reading is good. But getting your hands dirty in practice is better. The best type of learning is learning by doing.

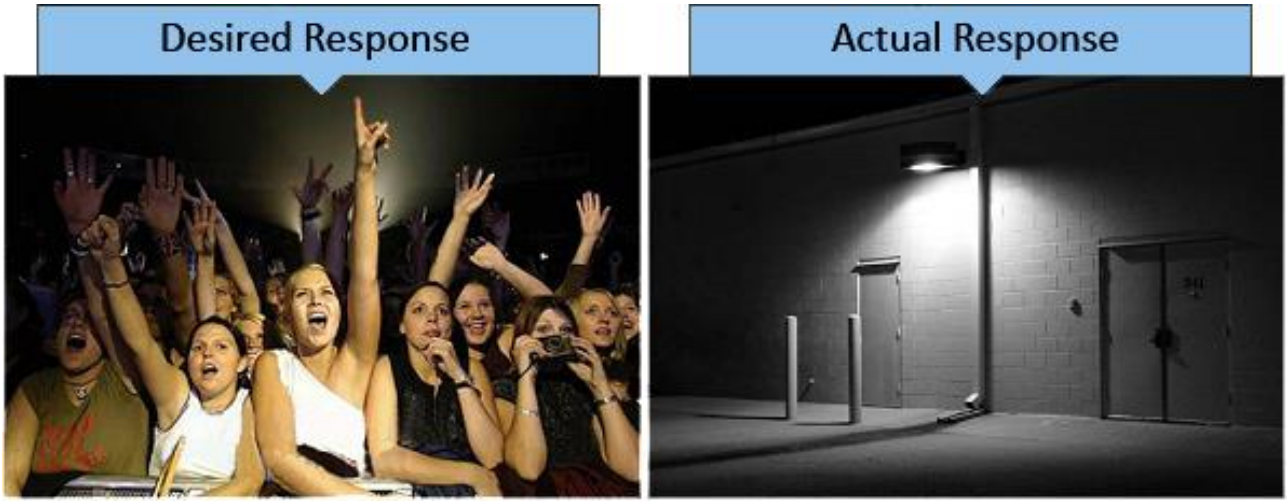
Personally, small projects and prototypes actually helped me improve. But there are still interesting things to master, therefore let's keep learning.

Should you build it if nobody comes?

By Darren Mart

Not long ago I received an E-mail from an energetic and talented developer who was tackling an ambitious software project. He asked if I had any advice to pass along.

It all came flooding back. The daydreams. Springing out of bed in the dead of night because an idea can't wait. Months of painstaking work, coaxing that baby from rolling to wobbling to walking on its own. All of this for an eager audience. Except...



The cycle after the cycle

I must have drafted a half-dozen responses to his mail. I admired his enthusiasm and understood where his head was. But I also knew what likely awaited him after the arduous development cycle: another cycle of turbulence, a series of clashes between expectations and unfortunate realities.

Where did it all go wrong?

This is a dangerous question we ask ourselves as developers. "Dangerous" because it carries the assumption that we did something wrong. It's entirely possible that what you've crafted is quite good, maybe even brilliant. The issue may have nothing to do with the quality of your work, rather your criteria for success.

So how does a developer accurately gauge the success of a solo project? I wish I had a concrete answer to that. Instead I can offer a few tips on how not to gauge it:

Tip 1: Beware the social platform sirens

Ah, but they lure us in with the promise of mass consumption and acceptance. We seldom consider their ability to trample our spirits. It's possible that you'll spend months on a project, share it on Facebook with the vigor and excitement of a dog with a chew toy, and then watch the "Like" count soar to... 3. One of those came from your mother, another from a page you made to promote your project, and the other from an errant tap on a cell phone.

It's the same chilling effect you get from photos of abandoned amusement parks. You strain to imagine giggles of delight and unbridled excitement, and you struggle to accept what's staring you right in the face.

Meanwhile, your friend's profound status of "I like ravioli!" earns 23 likes and 16 comments. Popularity and substance are two different things, and we'll just have to live with that.

Tip 2: The apathy isn't personal (it might not even be apathy)

You're already aware of this but it helps to be reminded. Most people have no concept of what it takes to pull off what you've accomplished. They don't realize you single-handedly built something that'd rival the efforts of an entire team. You can't, and won't, make them genuinely care.

Apps that set the world on fire generally fall under two categories: those that help users promote themselves, and those that require little thought. If your project requires a mental investment greater than that of Candy Crush Saga, here's the harsh reality: you're gonna end up with a lot of leftovers at your launch party.

Tip 3: Think twice before soliciting feedback from your peers

In the film Midnight In Paris, aspiring writer Gil asks Ernest Hemingway to read his novel and offer an opinion.

HEMINGWAY: My opinion is I hate it.
GIL: You do? But you haven't even read it.
HEMINGWAY: If it's bad I'll hate it because I hate bad writing and if it's good I'll be envious and hate it all the more. You don't want the opinion of another writer. Writers are competitive.

Don't overlook the profundity of this exchange. It applies to developers, too.

Tip 4: Like it or not, you're an artist

If you pour your heart and soul into solo projects because you enjoy the creative challenge, you're an artist. If you do it because you're attempting to solve a problem, you're probably an artist-engineer hybrid. If you do it strictly for the money, you bailed on this article a long time ago so it doesn't matter what I call you.

Just like the painters and writers and decorators and gourmet chefs of the world, we secretly hope the public will be elevated and inspired by our creations.

But there's a key difference. If someone isn't inspired by a painting, they'll still able to appreciate the individual effort. With software there's no such luxury. It doesn't matter if you're the Rembrandt of the coding world; your app is a dud compared to Office 365 or Google Maps or Skyrim, never mind the fleet of resources required by the latter.

So... should you build it if nobody comes?

Yes. Because as artists, we are enriched by the overall process regardless of the end result. Trite as it sounds, we have the sense of accomplishment and the pride of knowing how much discipline it took to see it all the way through. We stimulated our minds, we learned what does and doesn't work, we gained applicable experience. The next project, whether it's personal or professional, will reap the rewards.

Useful Job Search Websites For Programmers In The UK

By Necemon

Job boards are websites that facilitate job hunting.

There are many career websites designed to allow employers to post job requirements for a position to be filled; prospective employees can locate and fill out job applications and/or submit digital resumes for the advertised positions. Those sites may also offer employer reviews, career and job-search advice, and describe different job descriptions or employers.

As a technology specialist in the UK, here are some of the websites I found helpful in the past few years.

Graduates

targetjobs.co.uk
Graduate jobs and schemes. Internships and placements. Great advice to help you get hired.

eurograduate.com
Featuring thousands of graduate careers and job opportunities across Europe.

insidecareers.co.uk
Graduate jobs, internships, placements and school leaver schemes as well as career advice by sector.

milkround.com
The UK's most widely used student and graduate job website.

prospects.ac.uk
Prospects for graduate jobs, postgraduate study, advice about work experience, internship opportunities and graduate careers.

IT World

uk.dice.com
Formerly The IT Job Board. UK Contract and Permanent IT Jobs

purelyit.co.uk
Lots of contract and permanent IT Jobs including Software Engineer, IT Director, Senior Web Developer and many more.

computerjobs.com
Specialising in Permanent and Contract IT jobs in the UK with thousands advertised daily.

cwjobs.co.uk
One of the leading UK IT job board. Search information technology jobs and apply online.

currentitjobs.co.uk
Formerly findingitjobs.co.uk. IT job board focused on providing the best IT Jobs in London.

stackjobs.co.uk
A trending IT job board in the UK with effective IT recruitment solutions.

Science and Engineering

newscientistjobs.com
Technology and science jobs, courses and career advice.

naturejobs.com
Featuring access to job listings, editorial content about scientific careers and other information.

justengineers.net
A wide range of Engineering Jobs in the UK and Worldwide.

engineeringjobs.co.uk
Latest Engineering Jobs from across London and the UK.

General Job Search Sites

reed.co.uk
One of the leading UK job sites.

monster.co.uk
Possibly the most popular job site worldwide. Resources to create a killer CV, search for jobs, prepare for interviews, and launch your career.

jobserve.com
Powerful, quick job search. Specialising in permanent and contract jobs in the UK with thousands advertised daily.

jobs.ac.uk
UK & international job search for academic jobs, research jobs, science jobs and managerial jobs

totaljobs.com
Instant job matches, alerts and more from UK companies and recruiters.

topjobs.co.uk
Searchable database of vacancies by type and region.

neuvoo.co.uk
Presumably, your job search starts here.

jobs.trovit.co.uk
Job ads from thousands of websites in just one search.

jobbydoo.co.uk
Aggregating, analyzing and listing job openings from more than 1,000 sources, including UK career sites, job boards and recruitment agencies.

Microsoft Professional Certifications for Developers

By Necemon

Microsoft offers a wide range of online certification programs designed to help you grow your skills and your career.

This article will focus on developer certifications. Microsoft Certified Solutions Developer (MCSD) is a certification intended for IT professionals seeking to demonstrate their ability to build innovative solutions across multiple technologies. For example, the MCSD App Builder certification validates that you have the skills needed to build modern mobile and/or web applications and services.

I got my first Microsoft Certification back in 2008. I have been upgrading over the years and I am now a Certified Solution Developer. Was it worth it? Sometimes it didn't matter, sometimes it was pretty useful. In my experience, here are 5 advantages of getting certified:

- Getting a Microsoft Professional Certification doesn’t guarantee anything about getting a job, a raise or a promotion but it does increase the odds.
- When hiring someone new, some companies check out his certifications as well as experience. Many hiring managers verify certifications among job candidates and consider those as part of their hiring criteria. Some consider IT certifications a priority when hiring for IT positions.
- Getting a certification can boost your self-confidence, your confidence about your skills.
- It shows seriousness and passion (you did it all because you wanted to, not because you had to).
- The process of preparing for the certification increases your theoretical and practical skills.

Exam and preparation tips
My key recommendation is to dedicate a specific period to prepare right before the exam. Get some books, attend a course (virtual courses as a personal preference, but classroom courses available) and practice the technologies involved.
- Books: there are often preparation books related to the given exam. Example: Exam Ref 70-480: Programming in HTML5 with JavaScript and CSS3
- Classes : The most popular exams have short courses available online. Example: Course 20480B: Programming in HTML5 with JavaScript and CSS3
- Beyond the training material: researching the topics involved
- Actually practicing the technologies
- Answering to multi choice questions: Proceed by elimination. In case of doubt about the right answer, exclude the answers that don't make (any) sense, (or that makes less sense).

Career paths for developers
Make sure you double check that part. They keep updating the offers based on the latest technology but at the time of writing this, there are 5 options for MCSD (Microsoft Certified Solutions Developer)
- MCSD: Web Applications. Expertise in creating and deploying modern web applications and services.
- MCSD: Windows Store Apps. Expertise at designing and developing fast and fluid Windows 8 apps. There are two paths to achieving this certification-using HTML5 or C#.
- MCSD: SharePoint Applications. Expertise at designing and developing collaboration applications with Microsoft SharePoint.
- MCSD: Azure Solutions Architect. Expertise covering the full breadth of designing, developing, and administering Azure solutions.
- MCSD: Application Lifecycle Management. Expertise in managing the entire lifespan of application development.

Certification Planner
The Certification Planner is a tool to help you know what requirements are necessary to achieve your next certification. It's about planning your next steps (your options, which exams to take, in which order).

Other links of interest
MCSD Certification
Web Apps certification
Wikipedia Page

That's it for now. Now get back to work.

Final Reminder

Before we conclude, I just thought I would remind you that this document contains the Chapter 6 of the Album, and that 8 other chapters are also available. You may download, (re-)read or share any of the various chapters and volumes independently/separately, depending on your interests. The PDF, EPUB, MOBI/AZW3/KF8 (Amazon Kindle) and MP3 formats are available.

To get them, just click on any of the links you like below or go to TheAlbum.NeceMoon.com
(or necemonyai.com/Blog/page/The-Album.aspx)

The NeceMoon Album (complete)

Volume 1: Moon Light (softcore)

- Chapter 1: Strategy and Tactics
- Chapter 2: Digital Marketing and Web Visualisation
- Chapter 3: Corporate Worlds and Emerging Markets
- Chapter 4: Quick Wins, Tricks and Tips
- Chapter 5: Transition - Extra Thoughts and Sharp Fantasy

Volume 2: Full Moon (hardcore)

- Chapter 6: Software Development and Engineering
- Chapter 7: C# .NET Programming
- Chapter 8: Epic Prototypes, Classic Projects, Historic Genre
- Chapter 9: Research and Case Studies

The Album is available in French as well at Album.NeceMoon.com
(or necemonyai.com/Blog/page/L-Album.aspx)

Conclusion

This is Good Bye. However, we can stay in touch. Feel very welcome to add me on LinkedIn, Twitter or Facebook. If you also have any tools and tactics that help you achieve things efficiently, I would like you to tell me about that. My email address is necemon@gmail.com. You are more than welcome to write to me and report any possible mistake in this document, or to suggest any improvement, or to tell me about your favourite parts. However, if you don't like The Album, don't bother writing to me.

Anyways, I wish you all the best for your ongoing and next projects. Thank you for reading. And many thanks to Wikipedia, MSDN, IconFinder and FreeDigitalPhotos for the clarifications and the graphic resources. Special thanks to the Evasium Team. Thanks to all those who contributed, thank you Ahou The African Chick, Antoine Mian, Cyriac Gbogou, Darren Mart, Edith Brou, Holty Sow, Israel Yoroba, Jean Luc Houedanou, Jean-Patrick Ehouman, Karen Kakou, Nanda Seye, Nnenna Nwakanma, Olivier Madiba, Vanessa Lecosson and Yehni Djidji.
Thank you Monty Oum, rest in peace.

Share The Album with Your Mates

Just click on any relevant icon below. It only takes a couple of seconds and it's free for everyone. Alternatively, you can share this link on any social media website or email it to your contacts that would benefit from reading it: http://TheAlbum.NeceMoon.com