By John Sonmez
June 30, 2014

Step By Step Path to Becoming a Great Software Developer

I get quite a few emails that basically say “how do I become a good / great software developer?”

These kinds of emails generally tick me off, because I feel like when you ask this kind of question, you are looking for some magical potion you can take that will suddenly make you into a super developer.

I suspect that very few people who email me asking this question really want to know how to become a great software developer, but are instead looking for a quick fix or an easy answer.

On the other hand, I think there are some genuinely sincere developers that just don't even know how to formulate the right questions they need to ask to get to their desired future. I think those developers–especially the ones starting out–are looking for a step-by-step guide to becoming a great developer.

I thought I would make an attempt, from my experience and the best of my knowledge, to offer up that step-by-step guide.

Now, of course, I realize that there is no magical formula and that there are multiple paths to success, but I think what follows is a reasonable outline of steps someone starting out could take to reach a pretty high level of proficiency and be generally regarded as a good–perhaps even great–developer.

Step 1: Pick one language, learn the basics

Before we can run, we have to learn to walk. You walk by learning how to program in a single programming language. You don't learn to walk by trying to learn 50 million things all at once and spreading yourself way too thin.

Too many beginning programmers try and jump into everything all at once and don't have the patience to learn a single programming language before moving forward. They think that they have to know all the hot new technologies in order to get a programming job. While it is true that you need to know more than just the basics of a single programming language, you have to start here, so you might as well focus.

Pick a single programming language that you think you would be likely to base your career around. The programming language itself doesn't matter all that much, since you should be thinking for the long term here. What I mean is you shouldn't try and learn an "easy" programming language to start. Just learn whatever language you are interested in and could see yourself programming in for the next few years. You want to pick something that will have some lasting value.

Once you've picked the programming language you are going to try and learn, try and find some books or tutorials that isolate that programming language. What I mean by this is that you don't want to find learning materials that will teach you too much all at once. You want to find beginner materials that focus on just the language, not a full technology stack.

As you read through the material or go through the tutorial you have picked out, make sure you actually write code. Do exercises if you can. Try out what you learned. Try to put things together and use every concept you learn about. Yes, this is a pain. Yes, it is easier to read a book cover-to-cover, but if you really want to learn you need to do.

When you are writing code, try to make sure you understand what every line of code you write does. The same goes for any code you read. If you are exposed to code, slow down and make sure you understand it. Whatever you don't understand, look up. Take the time to do this and you will not feel lost and confused all the time.

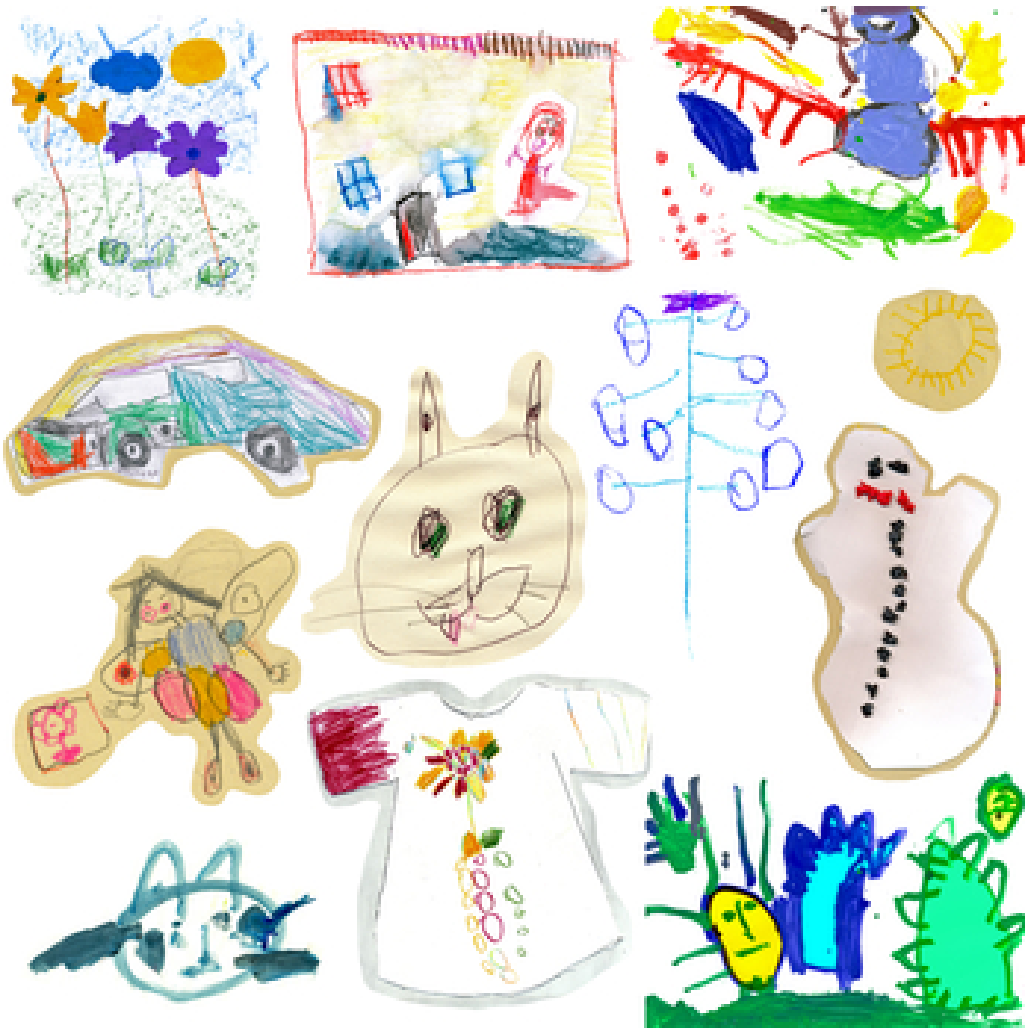
Finally, expect to go through a book or tutorial three times before it clicks. Y**ou will not get "programming" on the first try–no one ever does.** You need repeated exposure before you start to finally get it and can understand what is going on. Until then you will feel pretty lost, that is ok, it is part of the process. Just accept it and forge ahead.

Step 2: Build something small

Now that you have a basic understanding of a single programming language, it's time to put that understanding to work and find out where your gaps are. The best way to do this is to try and build something.

Don't get too ambitious at this point–but also don't be too timid. Pick an idea for an application that is simple enough that you can do it with some effort, but nothing that will take months to complete. Try to confine it to just the programming language as much as possible. Don't try to do something full stack (meaning, using all the technologies from user interfaces all the way to databases)–although you'll probably need to utilize some kind of existing framework or APIs.

For your first real project you might want to consider copying something simple that already exists. Look for a simple application, like a To-Do list app and straight out try to copy it. **Don't let your design skills stand in the way of learning to code.**



I'd recommend creating a mobile application of some kind, since most mobile applications are small and pretty simple. Plus, learning mobile development skills is very useful as more and more companies are starting to need mobile applications. Today, you can build a mobile application in just about any language. There are many solutions that let you build an app for the different mobile OSes using a wide variety of programming languages.

You could also build a small web application, but just try to not get too deep into a complex web development stack. I generally recommend starting with a mobile app, because web development has a higher cost to entry. To develop a web application you'll need to at least know some HTML, probably some back-end framework and JavaScript.

Regardless of what you choose to build, you are probably going to have to learn a little bit about some framework–this is good, just don't get too bogged down into the details. For example, you can write a pretty simple Android application without having to really know a lot about all of the Android APIs and how Android works, just by following some simple tutorials. Just don't waste too much time trying to learn everything about a framework. Learn what you need to know to get your project done. You can learn the details later.

Oh, and this is supposed to be difficult. That is how you learn. You struggle to figure out how to do something, then you find the answer. Don't skip this step. **You'll never reach a point as a software developer where you don't have to learn things on the spot and figure things out as you go along.** This is good training for your future.

Step 3: Learn a framework

Now it's time to actually focus on a framework. By now you should have a decent grasp of at least one programming language and have some experience working with a framework for mobile or web applications.

Pick a single framework to learn that will allow you to be productive in some environment. What kind of framework you choose to learn will be based on what kind of developer you want to become. If you want to be a web developer, you'll want to learn a web development framework for whatever programming language you are programming in. If you want to become a mobile developer, you'll need to learn a mobile os and the framework that goes with it.

Try to go deep with your knowledge of the framework. This will take time, but invest the time to learn whatever framework you are using well. Don't try to learn multiple frameworks right now–it will only split your focus. Think about learning the skills you need for a very specific job that you will get that will use that framework and the programming language you are learning. You can always expand your skills later.

Step 4: Learn a database technology

Most software developers will need to know some database technology as most series applications have a back-end database. So, make sure you do not neglect investing in this area.

You will probably see the biggest benefit if you learn SQL–even if you plan on working with NoSQL database like MongoDB or Raven, learning SQL will give you a better base to work from. There are many more jobs out there that require knowledge of SQL than NoSQL.

Don't worry so much about the flavor of SQL. The different SQL technologies are similar enough that you shouldn't have a problem switching between them if you know the basics of one SQL technology. Just make sure you learn the basics about tables, queries, and other common database operations.

I'd recommend getting a good book on the SQL technology of your choice and creating a few small sample projects, so you can practice what you are learning–always practice what you are learning.

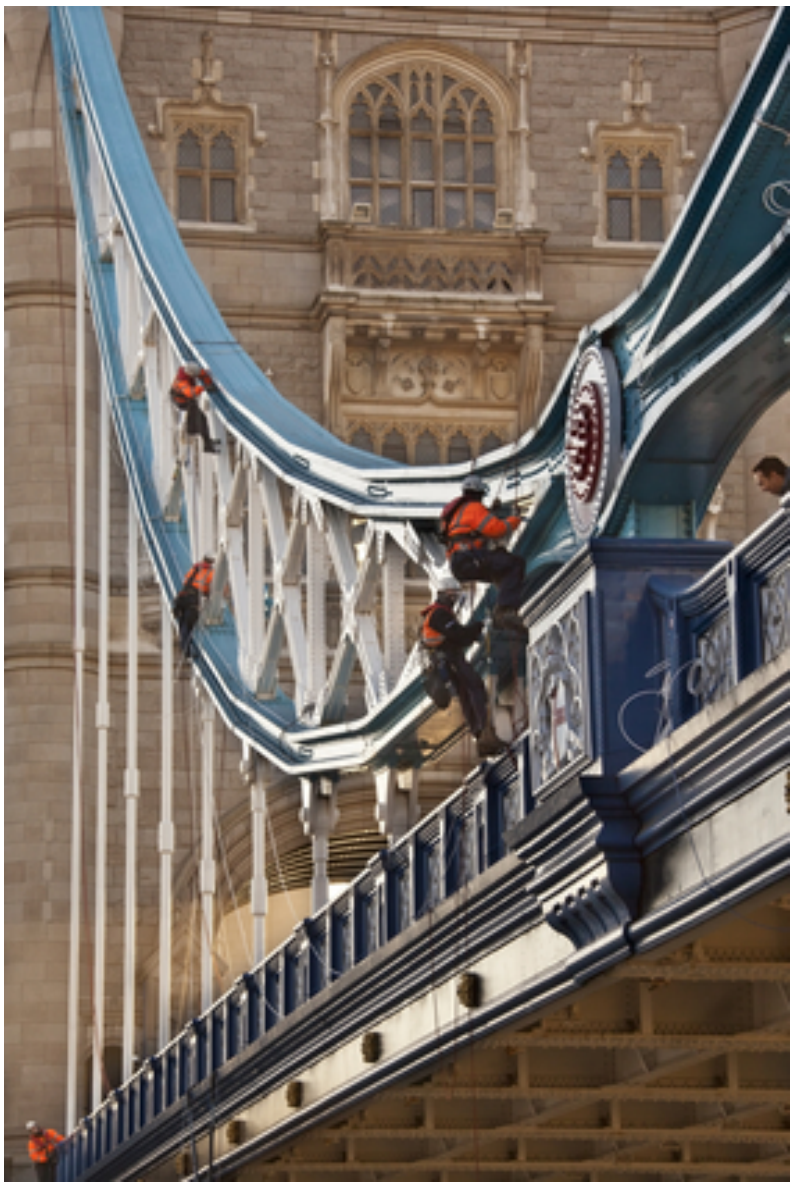
You have sufficient knowledge of SQL when you can:

- Create tables
- Perform basics queries
- Join tables together to get data
- Understand the basics of how indexes work
- Insert, update and delete data

In addition, you will want to learn some kind of object relational mapping technology (ORM). Which one you learn will depend on what technology stack you are working with. Look for ORM technologies that fit the framework you have learned. There might be a few options, so you best bet is to try to pick the most popular one.

Step 5: Get a job supporting an existing system

Ok, now you have enough skills and knowledge to get a basic job as a software developer. If you could show me that you understand the basics of a programming language, can work with a framework, understand databases and have built your own application, I would certainly want to hire you–as would many employers.



The key here is not too aim to high and to be very specific. Don't try and get your dream job right now– you aren't qualified. Instead, try and get a job maintaining an existing code base that is built using the programming language and framework you have been learning.

You might not be able to find an exact match, but the more specific you can be the better. Try to apply for jobs that are exactly matched to the technologies you have been learning.

Even without much experience, if you match the skill-set exactly and you are willing to be a maintenance programmer, you should be able to find a job.

Yes, this kind of job might be a bit boring. It's not nearly as exciting as creating something new, but **the purpose of this job is not to have fun or to make money, it is to learn and gain experience.**

Working on an existing application, with a team of developers, will help you to expand your skills and see how large software systems are structured. You might be fixing bugs and adding small features, but you'll be learning and putting your skills into action.

Pour your heart into this job. Learn everything you can. Do the best work possible. Don't think about money, raises and playing political games–all that comes later–for now, just focus on getting as much meaningful productive work done as possible and expanding your skills.

Step 6: Learn structural best practices

Now it's time to start becoming better at writing code. Don't worry too much about design at this point. You need to learn how to write good clean code that is easy to understand and maintain. In order to do this, you'll need to read a lot and see many examples of good code.

Beef up your library with the following books:

- Code Complete
- Clean Code
- Refactoring
- Working Effectively With Legacy Code
- Programming Pearls – (do the exercises)

Language specific structural books like:

- JavaScript: The Good Parts
- Effective Java
- Effective C#

At this point you really want to focus your learning on the structural process of writing good code and working with existing systems. You should strive to be able to easily implement an algorithm in your programming language of choice and to do it in a way that is easy to read and understand.

Step 7: Learn a second language

At this point you will likely grow the most by learning a second programming language really well. You will no doubt, at this point, have been exposed to more than one programming language, but now you will want to focus on a new language–ideally one that is significantly different than the one you know.

This might seem like an odd thing to do, but let me explain why this is so important. **When you know only one programming language very well, it is difficult to understand what concepts in software development are unique to your programming language and which ones transcend a particular language or technology.** If you spend time in a new language and programming environment, you'll begin to see things in a new way. You'll start to learn practicality rather than convention.

As a new programmer, you are very likely to do things in a particular way without knowing why you are doing them that way. Once you have a second language and technology stack under your belt, you'll start to see more of the why. Trust me, you'll grow if you do this. Especially if you pick a language you hate.

Make sure you build something with this new language. Doesn't have to be anything large, but something of enough complexity to force you to scratch your head and perhaps bang it against the wall–gently.

Step 8: Build something substantial

Alright, now comes the true test to prove your software development abilities. Can you actually build something substantial on your own?

If you are going to move on and have the confidence to get a job building, and perhaps even designing something for an employer, you need to know you can do it. There is no better way to know it than to do it.

Pick a project that will use the full stack of your skills. Make sure you incorporate database, framework and everything else you need to build a complete application. This project should be something that will take you more than a week and require some serious thinking and design. Try to make it something you can charge money for so that you take it seriously and have some incentive to keep working on it.

Just make sure you don't drag it out. You still don't want to get too ambitious here. Pick a project that will challenge you, but not one that will never be completed. This is a major turning point in your career. If you have the follow-through to finish this project, you'll go far, if you don't… well, I can't make any promises.

Step 9: Get a job creating a new system

Ok, now it's time to go job hunting again. By this point, you should have pretty much maxed out the benefit you can get from your current job–especially if it still involves only doing maintenance.

It's time to look for a job that will challenge you–but not too much. You still have a lot to learn, so you don't want to get in too far over your head. Ideally, you want to find a job where you'll get the opportunity to work on a team building something new.

You might not be the architect of the application, but being involved in the creation of an application will help you expand your skills and challenge you in different ways than just maintaining an existing code base.

You should already have some confidence with creating a new system, since you'll have just finished creating a substantial system yourself, so you can walk into interviews without being too nervous and with the belief you can do the job. This confidence will make it much more likely that you'll get whatever job you apply for.

Make sure you make your job search focused again. Highlight a specific set of skills that you have acquired. Don't try to impress everyone with a long list of irrelevant skills. Focus on the most important skills and look for jobs that exactly match them–or at least match them as closely as possible.

Step 10: Learn design best practices

Now it's time to go from junior developer to senior developer. Junior developers maintain systems, senior developers build and design them. (This is a generalization, obviously. Some senior developers maintain systems.)

You should be ready to build systems by now, but now you need to learn how to design them.

You should focus your studies on design best practices and some advanced topics like:

- Design patterns
- Inversion of Control (IOC)
- Test Driven Development (TDD)
- Behavior Driven Development (BDD)
- Software development methodologies like: Agile, SCRUM, etc
- Message buses and integration patterns

This list could go on for quite some time–you'll never be done learning and improving your skills in these areas. Just make sure you start with the most important things first–which will be highly dependent on what interests you and where you would like to take your career.

Your goal here is to be able to not just build a system that someone else has designed, but to form your own opinions about how software should be designed and what kinds of architectures make sense for what kinds of problems.

Step 11: Keep going

At this point you've made it–well, you'll never really "make it," but you should be a pretty good software developer–maybe even "great." Just don't let it go to your head, you'll always have something to learn.

How long did it take you to get here? I have no idea. It was probably at least a few years, but it might have been 10 or more–it just depends on how dedicated you were and what opportunities presented themselves to you.

A good shortcut is to try and always surround yourself with developers better than you are.

What to do along the way

There are a few things that you should be doing along the way as you are following this 10 step process. It would be difficult to list them in every step, so I'll list them all briefly here:

Teach – The whole time you are learning things, you should be teaching them as well. It doesn't matter if you are a beginner or expert, you have something valuable to teach and besides, teaching is one of the best ways to learn. Document your process and journey, help others along the way.

Market yourself – I think this is so important that I built an entire course around the idea. Learn how to market yourself and continually do it throughout your career. Figure out how to create a personal brand for yourself and build a reputation for yourself in the industry and you'll never be at want for a job. You'll get decide your own future if you learn to market yourself. It takes some work, but it is well worth it. You are reading this post from my effort to do it.

Read – Never stop learning. Never stop reading. Always be working your way through a book. Always be improving yourself. Your learning journey is never done. You can't ever know it all. If you constantly learr during your career, you'll constantly surpass your peers.

Do – Every stop along the way, don't just learn, but do. Put everything you are learning into action. Set aside time to practice your skills and to write code and build things. You can read all the books on golfing that you want, but you'll never be Tiger Woods if you don't swing a golf club.

Want more?

Sign up here to become a Simple Programmer. It's free and over 4,500 other developers have shown their support for making the complex simple.

ABOUT THE AUTHOR

John Sonmez

John Sonmez is the founder of Simple Programmer and a life coach for software developers. He is the best selling author of the book "Soft Skills: The Software Developer's Life Manual."

Related Posts

Working At Startups Vs. Big Companies As A Software Developer

JUL 25, 2019 / BY JOHN SONMEZ

The Modern Developer Part 1: Planning and Analysis

JUL 19, 2019 / BY ZLATIN STANIMIROVV

How To Become a Highly Paid Blockchain Developer

JUL 18, 2019 / BY GREGORY MCCUBBIN

Top Developer Collaboration Tools

JUL 15, 2019 / BY ROMAN ZHIDKOV

Teaching Yourself Machine Learning Without a MOOC or a Book

OCT 29, 2018 / BY ANDERSON ADDO

Write That First Blog: Blogging for Programmers

JUN 20, 2018 / BY SANDER ROSSEL

Top 50 Programming Interview Questions

APR 25, 2018 / BY JAVIN PAUL

Making the Transition from Manual Testing to Test Automation

FEB 10, 2017 / BY RYAN JOHNS

28 Comments Simple Programmer 1 Login

Recommend 3 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

T Pham • 5 years ago

Thanks a lot, M-John. As a "noob" programmer, this article will definitely be one of my most valuable guidelines to keep in my head.
Keep this up, M!

40 • Reply • Share ›

Bradley Ward • 5 years ago

Excellent article, John. And I totally agree with szelpe also.... learn to communicate; this includes presentations, release notes, etc.

Another skill that I think is important as you get into more and more senior roles is to pay attention to the business end of things. I'm not talking about selling or HR or that type of stuff. I'm talking about learning how what you do makes your company money, how it saves your customers money, etc. Young technologists often fall into the trap of thinking that without the technology, nothing else happens. I like to think that too, but the reality is that without a businessman willing to spend some money, nothing technological of any scale will happen. So learn how you fit into that.

Combine this understanding of how what you do affects the business with good communications skill, and you'll definitely move closer to the top of the pack.

2 • Reply • Share ›

jsonmez Mod → Bradley Ward • 5 years ago

Good point about the business end of things. I find it is actually pretty interesting as well.

• Reply • Share ›

Cà Rốt • 5 years ago

Thanhks John.

2 • Reply • Share ›

Peter Szel • 5 years ago

Hi John, nice list though I think you're missing a major part: the soft skills. I's great if you can code, but you need the learn to communicate, to present a presentation, to work in a team etc. Without soft skills you might become an awesome developer but no one will want to work with you so eventually you won't be successful.

2 • Reply • Share ›

jsonmez Mod → Peter Szel • 5 years ago

Yes. You are absolutely right. It is something to learn along the way. I'll add it to that section

Yes. You are absolutely right. It is something to learn along the way. I'll add it to that section.

1 • Reply • Share ›

Scott Nimrod • 4 years ago • edited

The Challenge that Changed how we Build Software...

Manager: Hey developer, I have a challenge for you...

Engineer: Yea boss? What's that?

Manager: I will offer you a 20 percent annual bonus at the end of the year, but under certain conditions.

Engineer: Okay! What do I need to do?

Manager: In order for you to earn your full bonus, your team must deliver on the following:

1. Your team's software never crashes or loses work after being handed over to QA.

2. Each bug discovered in your software must never be exposed again after it has been resolved the first time.

3. For each bug that's discovered by either QA or Production that directly conflicts with business requirements will result in 10% of your bonus being deducted.

see more

• Reply • Share ›

jsonmez Mod ➚ Scott Nimrod • 4 years ago

Simple: don't write any code. :)

• Reply • Share ›

Scott Nimrod ➚ jsonmez • 4 years ago

Really? You don't think greed is a strong driver for anyone who is extremely competitive?
What if I changed that 20% to $1,000,000?
Again, would your development habits change?
Mine would. And I already strive to automate quality into my code.
Philosophically, if greed drives the stock market, war, and government policies, which completely shapes our future by instilling new norms. I'm curious why it would not affect the way we build systems if the reward for doing so was so strong that it simply could not be ignored.

• Reply • Share ›

jsonmez Mod ➚ Scott Nimrod • 4 years ago

I'm saying that because, nothing in your rules said you had to produce anything.

I've actually written about what you are talking about:
http://simpleprogrammer.com...

It's an interesting thing to ponder.

Reply Share

• Reply • Share ›

Scott Nimrod jsonmez • 4 years ago • edited

"For each bug that's discovered by either QA or Production that directly conflicts with business requirements will result in 10% of your bonus being deducted."

I think the above requirement specifies that working software must be produced.

In addition, I think this idea could encourage developers to practice their Personal SoftwareProcess (i.e. http://en.wikipedia.org/wik....
I would hope that developers would take the lessons learned from their previous bonus deductions and code quality into their software to account for their past misses.

At the end of the day, developers are following a checklist of patterns of QA issues that cost the developer money.
Again, the end goal is not to be better at writing code, but rather, to get better with writing LASTING software.

• Reply • Share ›

jsonmez Mod Scott Nimrod • 4 years ago

True, good point. I missed that part. And, for the record, I agree with you.

• Reply • Share ›

Dustin Davis • 5 years ago

Nice write up, John. You did leave out an important task (at least explicitly), and that is to join and participate in the community. User groups, code camps, events such as hackathons, dojos, mob programming and so forth. That is a great way to meet developers and learn new ideas, topics and skills and it's access to valuable resources.

• Reply • Share ›

hal9k2 • 5 years ago

That was some great post, thank you John (I am at step 6 :))

• Reply • Share ›

Scott Fanetti • 5 years ago • edited

Another thing - be humble. Accept a code review not as a personal attack but as a learning experience. Being a developer you will never be the master of all domains. You must always be willing to accept that other people may know more or have a better understanding of some piece of code or some algorithm and its usage. Being open to criticism and not taking a line-by-line teardown of your code as a personal attack will both gain you cred as an honest dev - but will allow you to function effectively with the guys that don't have the soft skills mentioned below.

Often a new dev will run up against an ego driven superdev that is a master of the universe in his own mind. Being able to understand that person and learn what you can from him/her without taking their abrasiveness personally will help up-and-coming devs progress in their career.

I saw test driven development and behavior driven development - but for UI devs especially - the hallway paper test is essential to progress beyond the code monkey that sits in the cave to be the

CTO some day. Talk to the product folks and the sales folks to learn the domain of the systems on which you work. They don't know the code, but they know the world the code will live in. Try to see the customer as they see the customer and it will help you expand your ability to empathize and see the systems as a whole - not just the algorithms of this sprint.

Great article.

Reply • Share ›

jsonmez Mod → Scott Fanetti • 5 years ago

Thanks, and another great comment. Totally agree about the egos and code reviews. Thanks for the input.

Reply • Share ›

Scott Fanetti • 5 years ago

I have not read through all the comments yet - but learning to budget your time and properly estimate is also an essential task of a great developer. If you cannot understand the complexities of a system well enough to budget yourself spinup time you will fail. If you always overestimate and pad your time, you will never get enough done. Learning not to procrastinate and manage time is key to any successful person in any collaborative enterprise.

Reply • Share ›

jsonmez Mod → Scott Fanetti • 5 years ago

Very true. I agree.

Reply • Share ›

Ofer Zelig • 5 years ago

Excellent post, well thought, comprehensive.

Reply • Share ›

jsonmez Mod → Ofer Zelig • 5 years ago

Thanks

Reply • Share ›

Nhan Dao • 5 years ago

Thank you very much!

Reply • Share ›

Adedayo Samuel • 5 years ago

Great stuff John, and thanks for your honesty in laying it all down. Most articles offer a more general/vague approach & are not detailed and step-wise OR hide a few trade secrets. I really appreciate that.

I liked how you pointed out when to start with a 2nd prog language. I wanted to learn two simultaneously but I found peeps recommending starting with one but then I started wondering when the other one comes in and you answered that well. Arigato.

I most definitely agree with Bradley about adding some business intelligence, especially when you've attained some level of proficiency. I've come to find that businessmen get the better share of the pie. I didn't like/agree/understood why, but that seems to be the reality, at least from my

standpoint and I reckon it adds a lot of leverage to opportunities.

In all, great stuff once again. Thank you :)

Reply • Share ›

Thiago Ponte • 5 years ago

I've more than 3 years of software development, and i see myself around number 5, going to number 6.
What i see i miss doing is put things i learn to build something that's valuable.
This is a great list John, i'll make sure to come back to it from time to time!

Reply • Share ›

Jung Yi Lin • 5 years ago

Hello, May I translate this article to chinese and share it on my blog (http://wp.mlab.tw) ?
Thank you very much!

Reply • Share ›

jsonmez Mod ➚ Jung Yi Lin • 5 years ago

Yes. So long as you link back to this original post.

2 Reply • Share ›

Jung Yi Lin ➚ jsonmez • 5 years ago

Thank you so much. I believe your great article will encourage many Taiwan students.
The translated post is at http://wp.mlab.tw/?p=1869

1 Reply • Share ›

Bradley Ward • 5 years ago

Hi John. Don't know if I can post links here or not, but I read this article this morning and think it was pretty good too.

http://henrikwarne.com/2014...

Reply • Share ›

Sirwan Afifi • 5 years ago

Great, Thanks John.

Reply • Share ›

Subscribe Add Disqus to your siteAdd DisqusAdd Disqus' Privacy PolicyPrivacy PolicyPrivacy

← Previous post Next post →

Copyright 2018 by Simple Programmer. - Designed by Thrive Themes | Powered