How to Become a Programmer

Everything (Non-Technical) You Need to Know to Start Making Money Writing Code

www.SoftwareByRob.com
Rob Walling
v1.0

Contents

Introduction

Thank you for supporting this effort to provide battle-tested answers on how to become a programmer. I've wanted to write this book for at least three years, and the backlog of questions I received during this time makes this book more valuable than it would have been when I originally envisioned it.

Motivation

A question you might be asking yourself is, "Why would a programmer write a book about how to become a programmer?" Like every good question the answer is much longer than you probably care to hear. But here goes…

I receive a steady flow of emails asking some variation of the question, "How can I become a programmer?"

In early 2007 I put my thoughts down in abbreviated form in the post Advice on How to Become a Programmer, but judging by the number of new questions I continue to receive (all of which are answered in this book), it's obvious this subject deserves a more in-depth look than a blog post can deliver.

Everything contained in this book is based on 10 years of professional software development. I've been a salaried developer, a freelance developer, a consultant, a development manager, owner of a consulting firm, owner of a small shrink-wrapped software company (a.k.a. a MicroISV), and a technical author. I've written software that has never seen the light of day, and software that processes millions of transactions per day.

My blog is read by about 20,000 people each month, 80% of them software developers.

Updates and Questions

This book will be updated every few months as new questions surface. If you purchased this book from Software by Rob you are entitled to a lifetime of free updates.

If you didn't purchase this book from Software by Rob, please consider doing so. The proceeds contribute to updating and expanding this work, as well as answering emails from new and potential programmers.

If you have a question I haven't answered in these pages, feel free to email me at rob@softwarebyrob.com. I can't promise I will respond to every email, but it is very likely you will hear from me, if for no other reason for me to say "I've answered your question in the new version of the ebook, take a look."

What is Computer Programming?

If you've never written software before, the topic seems like a huge, foreboding black hole. When I tell people I'm a software developer I typically receive blank stares followed by a change of topic.

But it's really not so complicated. Computer programming, also known as programming, coding, or software development (there are subtle differences between each term), is the process of writing instructions that a computer can execute.

As an example: if you've ever written an Excel macro, you could say you've programmed a computer.

If you've ever built an HTML page you could say you've written code (some would argue with this, but it fits the definition).

And if you've ever written code in a mainstream programming language like BASIC, Pascal, C, Java, JavaScript, or PHP, you've definitely been inaugurated into the world of computer programming.

Why is Programming Necessary?

Programming is needed because computers are not very smart. Without someone telling them what to do they are nothing more than a hunk of hardware and some flashing lights. Software is what runs on top of the hardware, and it's where the really interesting things happen (says me, a software developer).

A computer program, also known as an application, or computer software, is a collection of instructions, or "code," written by one or more programmers and then executed on computers.

The code can be written in any one of a number of computer languages (a.k.a. programming languages), such as C, C++, Java, Visual Basic, Perl, PHP, etc… There are literally hundreds (thousands?) of programming languages, each with its own niche. But only a handful are widely used at any given time, and languages tend to last for a few years and then slowly slip away as new languages replace them. Some languages, however, stick around for decades (COBOL, C, and C++ are examples of long-lasting languages).

The computers that this software runs on can be desktop computers internal to a company, external customer computers, web servers, or other lesser-known specialized computers that run in cars, elevators and your microwave.

Microsoft Word, and the entire office suite, is a program that runs on many of our desktop (or laptop) computers.

Amazon.com's website is a massive, custom piece of software called Obidos that runs on thousands of Amazon web servers all over the world. When you open Firefox or Internet Explorer, it sends a request to one of these web servers, and an instance of Obidos replies with the information you request.

There's More to It Than Just Coding

While programming can be described as the basic act of "coding," or more simply described as "typing computer instructions into an editor," being a programmer is not just about writing code.

Software Development, a term you've probably heard used interchangeably with computer programming, is an umbrella term that encompasses the process of going from concept to finished product. While programming is the act of typing instructions, software development is the entire act of creating software, with a multitude of tasks, including:

- Speaking with users (or potential users) of your software to generate new feature ideas
- Writing spec documents to describe how software will function
- Discussing features and design approaches with other programmers
- Writing code (i.e. programming)
- Writing tests
- Testing your code
- Fixing bugs
- Preparing software for release
- Releasing it to a production environment or to manufacturing, where it will be distributed to its users
- And others…

While the act of programming is the most enjoyable part of the job for many programmers, numerous steps are required before and after you sit down at your computer and pound out a few thousand lines of code.

Why Should I Become a Programmer?

Programming (which I will also refer to as software development throughout this book; even though they are not identical I will use them interchangeably), is a challenging, high-paying career. If you have up to date programming skills, jobs are abundant. Programming jobs often come with flexible hours, location flexibility (work from home, a coffee shop, or another city), and the hard skills you learn translate easily to new positions.

Jobs

According to the US Department of Labor, 8 of the 10 fastest growing occupations between 2000 and 2010 are computer related. It seems like every magazine career survey I see has multiple computer-related jobs in the top 10. And, of course, there's the perpetual shortage of qualified programmers everyone has been talking about since the 1990s.

The result? There are tons of programming jobs.

A search on Dice.com will show you the literally thousands of open programming jobs (as of this writing, over 20,000 nationwide using the term "Developer.")

In my entire career as a salaried employee, I never went a day without work. Of course, your mileage may vary, but if you live near a major city it is almost certain there are hundreds of job openings at any given time for experienced programmers.

One example is a friend of mine who is a senior .NET developer in Los Angeles. He activated his resume on Dice a few months ago and received 40 phone calls in three days. There was so much interest he had to deactivate his resume to get any work done.

Hard Skills

When someone in marketing or sales looks for a job, it often takes months to land a position. When a developer with current skills looks for a job, they will often have an interview within a few days. This is in part due to the abundance of programming jobs, but also to the fact that programming is a "hard skill."

Hard skills are skills like programming, accounting, and engineering. Soft skills are skills like sales, marketing, people skills, etc… Both are valuable, but since hard skills are easier to communicate on a resume, and to test and quantify, the job hunting process is often much faster for programmers than for a product manager.

I guarantee that if you posted your resume on any of the major job boards with Senior .NET Developer, Senior Java Developer, or Senior PHP Developer in the title (with the experience to back it up), you would receive your first phone call in a few hours.

High Paying

In my opinion, you should never take a job simply for the paycheck. But if you enjoy programming (or think you will enjoy it), *and* someone will pay you top dollar to do it…that's not a bad way to go.

Take a minute. Go to Salary.com, type in Programmer 1 and your zip code in the salary wizard and take a look around. Though it's not the most common nomenclature, they use "Programmer 1" to represent a programmer with 0-3 years of experience. Programmer 2-5 represent more senior level developers.

In my zip code (Boston), entry level programmers are listed between $46k and $75k, with the majority around $60k. Not bad for 0-3 years of experience.

It's not uncommon to make $125k-$150k as a senior developer in a major city. This would require 5-7 years of experience and senior-level knowledge of a specific technology.

Since people with these skills are more difficult to find, it's actually easier to land a senior developer position than an entry-level one.

Flexible

I've worked from home 6 out of the last 10 years. The 4 years when I worked out of an office were when I was managing people, or when I worked from an office part of the week. A motivated programmer will get more done at home than at work because of the lack of interruptions. Progress is also fairly easy to measure, so managers tend to let programmers work from home, at least on an ad hoc basis.

For 5 of those 6 years I lived in a different city than my employer/clients. Becoming a programmer has allowed me to move from Sacramento to Los Angeles to New Haven to Boston, and I never once left a job because of geography.

All you need is high speed internet and a phone. I've written thousands of lines of code from coffee shops, the mountains, and even the beach.

Challenging

While the reasons I give above are appealing from an outsider's perspective, once you become a programmer it's all about loving your work. Some days you will sit in front of a computer for 10 hours, plugging away at a single web page…you have to love it or it will drive you nuts no matter how much money you're making.

If you know a programmer, ask if they enjoy their work. They will probably say some variation of the following:

"It's challenging, and I'm constantly learning new things."

The challenge is in solving problems. Some problems are as simple as "How can we give this customer a new button on this page so they can print from here?"

Others are as complex as "We have three years of code written in VB.NET and we want to transition to C#. What do we do?"

But in the end, every person I know who loves programming loves solving problems. Programming truly is a new problem every day.

The Act of Creation

Forgive me, but I'm going to get a little philosophical on you. Some people won't care about this reason, but it's one of the primary reasons I like coding so much.

There is a lot of discussion about whether programming is art or science. I believe it's a bit of both, but that it doesn't totally matter.

What matters to me is the rush of endorphins I feel when I hammer out a bunch of code and demo something for the first time. The first time a page pulls records from a database and displays them, I'm filled with a huge sense of accomplishment. It sounds stupid, I realize, but the fact that I've created something tangible (i.e., a web site) is perhaps the biggest reason I have continued to write code for the past 10 years.

Creating something from nothing is a great feeling.

It's Sometimes Cool

Ok, programming is not usually considered cool. Many times you'll be building an invoicing system for an internal application that only a few people will use. It's not as boring as it sounds, but it's no trip to Disneyland, either.

Then there are times when you build the web site for the Sundance Film Festival (I did in 2003), or work for MySpace or YouTube on a high profile feature (like friends of mine have). Or a site you work on sells $17 million worth of product in a year (which has happened to me and a few friends).

During those times it's fun…and yes, maybe even cool to be a programmer.

What Are Some Reasons Not to Become a Programmer?

If You Don't Like Programming

If you're a good developer, you're going to be living in code 40+ hours a week (and probably thinking about it on the weekends). If you don't like the act of programming, this job is not for you.

You Have to Constantly Update Your Skills

Programming languages and technologies change every 2-3 years. If you do not have a thirst for ongoing learning, programming is not for you. Good developers are lifelong learners and continue to read books forever.

Management

It's hard to find good managers in any area of a company, but it seems that the situation is especially bad in Information Technology (IT). Many IT Managers were once techies themselves, so they are predisposed towards technology rather than people. Nowhere have I seen more poorly run teams than in IT.

Of course, this could also be motivation for some people to become developers so they can one day manage them well. There is definitely a lot of room in the business world for good IT managers.

Overtime

It varies from situation to situation and week to week, but since programming is deadline driven, at one time or another you're going to have to work overtime. Overtime for programmers is almost never paid. Some jobs require it on an ongoing basis, most jobs require it every month or two.

What are the Different “Worlds” of Programming?

Different people have their own takes on the different “worlds” of software. Joel Spolsky, of popular blog Joel on Software, defines them a bit differently than I, but the general idea is the same. Here is how I define the six worlds of software:

Products

This includes working for a big company like Microsoft on Word or Excel, for a company like Google on Google Maps or Gmail, or for a smaller company like Salesforce.com on their namesake web application.

Product development is extremely challenging, often with tight deadlines and a lot of overtime before a launch. I would venture to say that the most gifted developers move towards product development, since it poses the most difficult problems and allows you to build higher quality software than you can justify in corporate development.

Corporate Development

This typically looks like working for a bank, insurance company, or other big corporation developing software for their accounting department, call center, shipping department, etc...

In this scenario you will work with enterprise technologies like .NET or Java, and will build web applications, desktop applications (typically .NET Windows Applications), or mobile applications.

I was a corporate developer for several years of my career. This is where the majority of software jobs are, and the pay is high compared to the amount of experience you need.

Embedded Software

Embedded software is burned onto a custom chip and includes software that runs in your car, elevator controllers, and handheld GPS devices.

I don’t know much about this area, other than that it’s more like building hardware than software, since you get one shot to get things right (there are no re-releases once it does out the door). The release and testing cycles are long because there are no second takes. I see embedded job openings from time to time, but in my (very limited) experience, it’s a niche arena.

Writing embedded software can be very challenging and you can make great money, but your job options are more limited than with corporate development.

Game Development

If you’re into games, building them is a blast. It’s also a huge amount of work.

The first-hand accounts I've heard from game developers indicate they love the job, but hate the long weeks (60-80 hours). The pay also tends to be lower than corporate development, which makes sense? If you were building an invoicing application wouldn't you take a pay cut to work on Halo 3?

Game development requires a mathematical mind, and is the one area of development where I say that college-level math is a necessity.

Consulting

You can consult for a big company like EDS, BearingPoint, or Accenture, or for a small consulting firm (which I did for several years). Consulting is a lot of fun, and you tend to work with cutting edge technologies and new coding techniques. But working for big consulting companies requires a lot of travel, and small ones will probably not have full-time work for you.

The pay tends to be comparable or higher than corporate development. This was my area of choice after being a corporate developer and before I started my own consulting firm.

Freelance

Once you have experience and contacts you may start doing projects on the side, and may eventually transition into a freelance developer. The money is great, but the hard part is maintaining a steady stream of work.

You get to work for yourself and have all the freedom of being your own boss, but you have to buy your own health insurance, don't get paid when you take a day off, and you have to handle your own marketing, sales, invoicing, retirement savings, etc…

But this is an attractive and attainable goal for many entrepreneurial-minded developers, myself included.

What Programming Language Should I Learn?

This is a fairly subjective question, much like asking "Should I learn Spanish, French or German if I want to be an interpreter?" It's difficult to answer without knowing something about what you'd like your career to look like once you become a programmer.

Because you're likely reading this ebook with minimal knowledge of programming languages, I don't want to throw out 25 language possibilities for you to learn.

There are groups of languages based on the Software Development Worlds I listed above. In general (note: I'm making sweeping generalizations and choosing the most popular languages simply to cut down on confusion):

- Desktop products are typically written in C++, C#, VB.NET, or Java
- Web applications are typically written in ASP.NET, PHP, Ruby on Rails, or Java
- Corporate development typically uses either the Desktop or Web languages, or both
- Embedded software typically uses a specialized version of C or C++
- Game development typically uses C++ (and more recently, C#)
- Consulting and Freelance work tend to be done in the Desktop and Web languages

Since there is a lot of language crossover between the worlds, I'm going to assume that one of your higher priorities is to get a job in most developed parts of the world, and that you want to invest your time into learning a language that will continue to show job growth over the next several years.

Following that logic and the opinion of many people in the software industry (including me), desktop application development is dying. It's not dead yet, but for the most part, web languages are the direction to head for future growth. Web programming is a growth industry, and it's going to be around for a long time.

With that in mind, here are the major web languages:

ASP.NET

ASP.NET is my platform of choice (C# and VB.NET to be specific). ASP.NET runs on Windows and the IIS web server (included with Windows). You use Visual Studio to develop, and typically use a SQL Server database ((free versions of both tools are available).

This is the route I would recommend only because it's easy to start developing if you have a Windows computer, and it allows you to work for large enterprises, small companies, web product companies, or yourself. There are many, many .NET jobs available and .NET is one of the most popular programming platforms in the world.

PHP

PHP is an open source scripting language, and one of the only other languages you should consider. It's great for building websites using free tools. It runs on the Apache web server, and typically uses a MySQL database (both are open source).

There are many PHP jobs out there, but not as many high-paying jobs as .NET developers. As an aside, enterprises are more likely to use .NET or Java, while startups are more likely to use PHP or Ruby on Rails.

Ruby (on Rails)

Ruby on Rails is a cool, new web 2.0 framework. Ruby is the programming language, Rails is a framework that allows you to build web applications super fast. It's not being taken seriously by too many large companies, but due to the speed of development, startups are using it left and right.

Java

Java is an open-source enterprise language that's owned by Sun. You can develop desktop or web applications using it, but Java is losing ground to PHP and Ruby on Rails on the startup front, and .NET on the enterprise (large company) front.

ColdFusion

ColdFusion is owned by Macromedia (now Adobe). It's more for designers, but it's an easy scripting language that integrates well with Java. I wouldn't bother learning it; it's been on its way out for a while.

Perl

Perl is an old open source text parsing language. It's still used here and then, but it's much more of a language for back-end maintenance scripts. You'd be nuts to build a new website using it these days.

If being gainfully employed is your goal, becoming a web developer is, in this developer's opinion, the best approach. There are tons of ASP.NET, PHP and Java jobs out there (and more Ruby on Rails jobs every day).

Ruby on Rails development is hot (and also a lot of fun). If I were starting out I would build a crazy cool project in C#/ASP.NET or Ruby on Rails, and leverage that to get my first programming job. Before you decide on which language to learn, look at Dice for your area and figure out which languages are in high demand, and what type of company you'd like to work for.

If you know for sure that you would rather become a game programmer, learn Microsoft's XNA, which is an introductory platform for making games that run on Windows and the Xbox 360. As a game programmer you will eventually need to learn C++, but I would not attempt to learn it as my first language.

Where Do I Start?

So you know what programming is, and why you might want to become a programmer. But you've never written a line of code (or maybe you have, but not since you were in high school or college).

I'm a self-taught programmer, but I also earned a degree in Computer Engineering, so I've seen both sides of the coin. Learning programming from a book teaches you the practical side of how to write code. Learning it in college teaches you the theory of why we do it the way we do.

With that in mind, if I were getting started today, I would follow these steps:

1. Buy Beginning ASP.NET 3.5: In C# and VB and read it cover to cover, doing all of the programming exercises inside. When you are done you will have a good handle on how to build a basic ASP.NET web application.
2. Pick a small project and build it for yourself. Examples include: an online replacement for a thumb drive, a web-based contact manager, or a time tracking application. Don't worry about how it looks, for now worry about how it functions. Code it up and test it out.
3. Once it's complete go back and work on how it looks.
4. Upload it to a hosting account and make it "live" on the internet. You now have .NET programming experience with an application that's live on the internet and a showpiece for potential employers.

This is one of thousands of paths you could follow to gain programming experience. I chose .NET because there is a huge demand for .NET developers. If you think you would prefer to develop in PHP, buy an introductory PHP book like Beginning PHP and MySQL: From Novice to Professional and follow the same steps.

Variations of the above hold for game programmers or desktop programmers; the steps stay static, but the book and sample project vary.

The above steps accomplish a number of things:

1. They get you writing code. Nothing is more worthless than reading about coding for 6 months. You have to start writing it as soon as possible.
2. You will learn a marketable language. Real applications are built using ASP.NET and PHP. You will be learning a language that will probably get you a job in a few months.
3. You will create a portfolio piece. It won't be the sexiest thing you've ever seen, but it will be an application you can demo to a potential employer, and will include sample code you can show them.

4. By the end of step 2, you will know if you like programming. Most programmers I know would not be able to put the book down, and would think about the sample project day and night until it was completed. If you find yourself dreading sitting down at the computer and spending the time it takes to build your project…it's possible that programming is not for you. Of course, try to separate your tiredness from work or lack of time as possibly de-motivating factors.

I’ve Built a Project and Decided I Like Programming, What Next?

There are a number of possibilities for expanding your programming knowledge:

Learn While Doing

Do you want to *really* learn to code? Get a job writing it. Even if you only make $10 an hour; you will progress more in 1 month as a full-time developer than you will in a year of hobby programming. There’s no better way to learn to program than to do it.

Books

Books have been critical in my quest for programming knowledge. When I’ve apprenticed developers in the past, I use books as their primary source of basic knowledge, having them read 1-2 programming books per month (see the resources chapters at the end of this book), while teaching them more advanced techniques in person.

Online

The web is a good resource for answering programming questions, but I’ve found that when learning new concepts from scratch I need something I can read and digest. If the web works for you, great. Sites like http://www.php.net, http://www.asp.net, and http://java.sun.com/ are the places to start for their respective languages.

A Software Apprenticeship

If you haven’t read my article on Software Apprenticeships, I recommend you do. The best (and I would argue the quickest) way to become a good programmer is to write code under the wing of an experienced developer who will teach you not only the basics, but the in-depth knowledge that takes years of experience to learn. I consider this option leaps and bounds above the others. The trick? Finding a programming apprenticeship is very difficult.

College

Having gone this route myself I am well aware of the limitations of the University system in preparing students for a career in computer programming. Preparing them for a career in determining little-o and big-O notation, sure, but actually writing code from the get go? Nope.

College is great for high-level theory (which you will want as you become a more senior developer), but work experience trounces it in the first few years of becoming a software developer.

Tech school

I’ve only worked with one programmer who went to a technical school and she had good knowledge of language and coding concepts, but not a ton of theory or design knowledge. As a result, her code was utilitarian and used a lot of brute-force, but was often not well-designed or easily maintainable. There’s obviously a balance between not enough practical knowledge (college) and not enough theoretical knowledge (tech school). I am using a very narrow sample, so don’t take this as a blanket judgment of tech schools. As an aside, the network administrators I’ve worked with from tech school have been well-trained and great to work with. Perhaps the nuts and bolts of networking are better suited for such a practical teaching approach.

I Just Graduated from School, How Can I Get Experience?

Above everything else, the best way to get experience writing code is to get a job writing code. For learning, you can't beat writing code 8 hours per day and getting paid for it.

But if you can't find an entry level position right off the bat, while you're looking you should be building projects on your own, or contributing to open source projects.

Building Projects on Your Own

A key thing that sets programming apart from other engineering disciplines is the fact that people do it as a hobby. No one designs mechanical systems for fun. But thousands of people write code in their spare time purely for the challenge and the satisfaction they get from creating something.

Building something on your own is not as scary as it sounds. With a little programming knowledge learned from the previous section, you could have a simple web application built in a matter of days. Every piece of experience you pick up from this time will be critical once you land a job as a developer.

Getting a job with no professional programming experience is not nearly as hard as it sounds. Getting a job with absolutely no programming experience is.

While professional development experience is ideal, you have to start somewhere. If you apply for an entry level position and you've built at least one sample project that is live on the internet, you will stand head and shoulders above the competition. I can't tell you how impressed I've been with recent graduate who showed the initiative to not only teach themselves a modern language, but spent the time to code a project and put it online.

"Hire" instantly came to mind.

Open Source Projects

From Wikipedia:

> Open source software (OSS) began as a marketing campaign for free software. OSS can be defined as computer software for which the human-readable source code is made available under a copyright license (or arrangement such as the public domain) that meets the Open Source Definition. This permits users to use, change, and improve the software, and to redistribute it in modified or unmodified form. It is very often developed in a public, collaborative manner.

This means that anyone, even a new programmer, can contribute code to an open source project. You may have heard of Linux, Apache, WordPress, PHP, and Java; all are open source software projects.

Go to http://sourceforge.net/ or http://www.codeplex.com/ and take a look around. It's daunting at first, but there are tons of projects that could use your help. Pick a project that sounds interesting and read how you can get involved. You may need to apply for access, or you may be able to check changes in right away.

Either way, adding "Open Source Contributor" to your resume will be a huge advantage.

How Can I Become a Programmer Without Going to College?

Around 60% of Web Developers have four-year degrees, and according to the U.S. Bureau of Labor, 68% of Computer Programmers (an umbrella term that includes Web Developers) have four-year degrees.

So can you become a programmer without a college degree? Absolutely.

3 out of 10 programmers and 4 out of 10 web developers do not have four-year degrees. So while the odds are not as favorable as if you went out and got one, if you don't already have a degree it's not worth it just to find out if you like programming.

The things I look for when hiring someone who will work as a developer are (in approximate order of importance):

1. Professional Experience
2. Certifications
3. College Degree
4. Contributions to Open Source Projects
5. Articles/Publications, Speaking Engagements, A Popular Blog
6. Hobby projects

On a side note, few things are worse than a developer with certifications and no professional experience.

Before you go out and pay a huge amount of money to get certified in a technology you don't understand, learn to write code. Build your sample project, and then think about contributing to open source projects. Finally, think about getting a certification.

Depending on the size and nature of your sample project, and on your competition for an entry level position, your sample project may qualify you for the coveted #1 spot given that your competition is not likely to have any of 1-5, either, unless they have a degree.

Although entry-level programmer positions are competitive, I've known many good programmers who never went to college, and worked their way into programming through hobby projects, 2-year degrees, or transferring from other areas within IT (such as the help desk). These are also good approaches to think about when plotting your entry into programming.

What About Offshoring – Will I have a Job in 5 years?

If you've read Thomas Friedman's book The World Is Flat: A Brief History of the Twenty-first Century, you'll know that the hype of offshoring, that all of our programming and call center jobs are being sent overseas, is not entirely true.

The reason offshoring has received such media attention is that it makes a great story, and there are numbers to back it up. The problem with the attention is that it's pretty one-sided. The papers never show how many jobs are being created back in the U.S. due to offshoring. They also tend to shy away from the still-dire need for skilled programmers we have in the U.S.

If you followed Dell Computer's foray into offshoring they call centers, you may have heard that they:

> …have even pulled some of their call center work back to the U.S. after complaints from customers about everything from foreign accents to the quality of support.

Companies have discovered that although it's less expensive on the front-end to offshore jobs, it's not always a true net gain once you deal with the time difference, communication differences, lack of physical meetings, and other issues that have caused some to pull back in their offshore ambitions.

That's not to say that offshoring is not a very real thing that's happening in our economy, but the fact is that very few white collar jobs are safe from offshoring. Accounting, electrical engineering, mechanical engineering…all are being offshored to one extent or another. Unless you plan to become CEO, an administrative assistant (although virtual assistants are taking over there, as well), or take on a job where you have to be physically present, you're going to run the risk of your job being sent overseas.

The unemployment rate for computer and mathematical occupations was 37% lower than the overall U.S. unemployment rate in 2007 and 43% lower in October 2008, even given our current financial crisis. Also, check out this graph, showing data from 2000 to 2004, with the trend of Computer Occupation unemployment substantially below overall U.S. unemployment.

We all know the world is in financial crisis, yet I've received 3 emails this week from recruiters asking if I know of .NET developers to fill their open positions. And I started a new project for a client who just received funding last week. Indeed, there is always programming work to be had.

Is offshoring real? Yes. But until we've perfected teleportation, there will always be work for skilled developers.

Do I Have to Know Math to be a Programmer?

It’s not necessary to know math to be a programmer.

But as I said above, if you’re developing games, mathematics and physics play a large part and college level math is a requirement. But building an invoicing application does not require much beyond basic addition and subtraction.

With that said, many of the concepts in programming stem from algebraic thinking.

And from my personal experience, people who have an easy time learning mathematics and enjoy solving logic problems (be they algebra or puzzles from how to move Mount Fuji), tend to learn code quicker and enjoy it more in the long run. It takes a twisted mind, my friends.

One thing I would say is that the more math you know the more interesting are the problems you can work on. But telling someone who has never written a line of code that they should go learn algebra would be poor advice.

How Much Money Can I Make?

I’ve never worked with a programmer, except one who worked for a non-profit, who made less than $40,000 per year. That is straight out of college, entry level corporate development.

The majority of entry-level coders I’ve worked with have made $50-$55k. Mid-level developers make $65k-$90k, and senior developers make $85k-$150k (and even more if their knowledge is highly specialized).

If you become a freelance developer you will make $50-60/hour as a mid-level freelancer, $60-$80/hour as a senior developer, and $80-150/hour as an expert in your field (i.e., you have a well-respected blog, publish articles, or have some other specialized niche).

These are ballpark ranges based on my experience, and based on corporate development salaries in larger tech centers (San Francisco, Los Angeles, Seattle, New York, and Boston). For ranges for your area, visit Salary.com and enter “Programmer” and your zip code in the Salary Wizard, or search on Dice for your zip code and look at the open job listings.

How Do I Become a Great Programmer?

Everyone I've talked to has taken a different path. Programming is different from other engineering disciplines; if you want to become an electrical engineer you go to school, graduate, work for an engineering firm, and one day take a test and get licensed. Programming is different because people do it as a hobby; no one designs electrical subsystems for fun. This creates more possibilities for learning how to code.

Here are the elements I think are critical:

Learn, Learn, Learn

You must have an insatiable appetite for knowledge. This usually means reading a programming book every few weeks in the early days, and moving on to more conceptual books like The Pragmatic Programmer, Code Complete, and Facts and Fallacies after 6-12 months of full-time coding. I can't stress enough the value of reading, or the value of immersing yourself in code early in the process.

Transition into Concepts

Learning how to be a good programmer begins with learning logic concepts and language syntax; they are much easier to understand when taken together. But good developers quickly desire knowledge that transcends language syntax. Perl, PHP, Java, ColdFusion, ASP…all languages I used in my first 18 months as a professional programmer. What made me a good programmer was not my knowledge of each language, but my desire to understand and refine concepts like D.R.Y., the broken window theory, and code re-use (all gleaned from one of the best books for becoming a good developer – The Pragmatic Programmer).

Hang Out With Programmers Who Are Better Than You

As a guitar player, the most I ever learned about the craft of songwriting was when hanging out with people who were much better than I. The same goes for writing software.

And due to this new-fangled internet thingy you don't even have to be physically present to be a part of the community: read programming blogs from the heavy hitters (Scott Guthrie, Rocky Lhotka, Dino Esposito, Scott Mitchell, etc…), check out programming forums, and look at other peoples' code.

Reading source code can be a pain, but the more you see the more you will be able to identify code that's easy to understand, and code that takes a PhD to figure out how they output "Hello World" to the screen.

And Finally…

This is one of the best comments on my blog from recent memory:

I would tell someone wants to be a programmer that the world has enough “programmers”. What we need, and what you want to be, is an “engineer.” I run into lots of “programmers”, folks who have read “Learn C# (or whatever) in 21 Days” and think they can program on 100k LOC systems. If I were guiding a young person looking to get into software, I would teach programming, yes, but only as a facet of software engineering:

- Learn how to gather requirements.
- Learn how to design a system based on the requirements, and look for design patterns, such as n-tier, MVC, etc…
- Write your code so that it is testable and maintainable.
- Learn to WRITE TESTS!!
- Learn to refactor.

Honestly, programming should be about half of what an engineer does.

Beginning Resources

Look at any programmers' bookshelf, and you will see a line of books covering a range of technical topics. Even with the resources available on the web, books are unsurpassed in their coverage of software development.

In general, programming books from Wrox, O'Reilly and Apress are high quality.

The books below are about learning language syntax and basic programming concepts. Choose the books for the language you are interested in pursuing based on the discussion in "What Programming Language Should I Learn?"

For game development, see the question "How Do I Make a Computer Game?" in the FAQ section, below.

ASP.NET

Beginning ASP.NET 3.5

Beginning ASP.NET 3.5 in C# 2008

Ruby on Rails

Beginning Ruby on Rails

Beginning Ruby on Rails E-Commerce

PHP

PHP 6 and MySQL 5 for Dynamic Web Sites: Visual QuickPro Guide

PHP and MySQL Web Development

Java

Head First Java

Beginning Programming with Java For Dummies

Advanced Resources

The books listed below cover concepts that transcend language syntax, and are for reading once you have a good handle on the technical side of programming (typically 6-12 months in).

The Pragmatic Programmer: From Journeyman to Master

Code Complete: A Practical Handbook of Software Construction

Refactoring: Improving the Design of Existing Code

Rapid Development: Taming Wild Software Schedules

Peopleware: Productive Projects and Teams

Software Requirements

Software Estimation: Demystifying the Black Art

Frequently Asked Questions (FAQ)

How can I become a .NET programmer?

Your best bet is to buy one of the books I mentioned above, read it cover to cover, and build your own dynamic website. As I've said, it's the fastest way to learn a language without paying thousands of dollars to attend a training course.

What is my best path to programming if my primary computer is a Mac (OS X)?

Since Mac's are now built on top of Unix, they run any Unix-compatible languages. The languages I discussed in this book that are compatible with Unix include PHP, Ruby, Java, and Perl.

Personally, I would learn Ruby or PHP. If you want to build games, go for C++ (but realize it's a very tough language to start with). A good book to start with is Sams Teach Yourself C++ in 21 Days.

I want to start a second career as a programmer. What is the fastest way to do it?

If you are in an existing career, think about how you might transition into programming at your current job. As an example, if you work in finance at a large company, it's very likely there are programmers in house who are writing software that you use. Make friends with the CIO/CTO and find out if they would be willing to bring you into the fold. You'll need to learn something about code first, and I would suggest learning from the books I mentioned herein.

You may also want to prove your technical chops by handling a lot of the more complex Excel and Access-related work, since these programs have overlap with more advanced programming and database concepts.

What are the simplest computer languages to start with when learning to program?

Visual Basic (now VB.NET) is simple to learn, and is still a powerful language. You could go back to languages like Pascal or BASIC, but these languages are not used in new development so you would be spending time learning syntax that will not help you get a job.

VB.NET is used when coding ASP.NET, so if you were to purchase Beginning ASP.NET 3.5, you would learn to build websites using both VB.NET and C#.

What language should I learn to become a systems programmer?

Systems programming is used to build software that provides services to your computer hardware. In other words, people do not use this software, your system uses it. Examples include a disk defragmenter or device drivers.

All systems programmers I know are aces with C and C++.

If I feel I am not pre-disposed to be a good programmer but I have the desire, should I still do it?

This is a tough question. My gut feeling is that you should give it a try to see if you like it. But if you don’t latch onto it early in the process, don’t force it. Programming comes with a lot of negatives, like bad bosses, potentially long hours, and sitting in front of a computer all day. You need to love it to do it well.

What would you recommend if I want to specialize in programming for peer to peer applications?

Peer to peer applications are applications that don’t use a central server, rather they communicate directly from one desktop application to another, like Napster and many instant messaging clients.

.NET or Java would suit you best for peer to peer programming. .NET Windows Forms (which you can write in C# or VB.NET) will run on any recent Windows computers. Java is a bit harder to learn from the start, but it runs on multiple operating systems (Windows, Linux, Mac and others).

What is Agile Development?

You may hear this term thrown around. The best definition I’ve seen is on Wikipedia:

> Agile software development refers to a group of software development methodologies. Agile methodologies generally promote: A project management process that encourages frequent inspection and adaptation; a leadership philosophy that encourages team work, self-organization and accountability; a set of engineering best practices that allow for rapid delivery of high-quality software; and a business approach that aligns development with customer needs and company goals.

In other words, with Agile development you don't write a big spec at the beginning of the project and stick to it – the application development process is much more fluid.

Should I plan on paper first before starting a project?

You should write a specification document, even a rudimentary one, before building a new project. But I would not recommend writing out your code on paper before entering it into the computer. I knew people in school who did that, and those days are long gone.

How can I make a computer game?

There are two routes I know of for game programming: the hard-core, legitimate route, which is to learn C++ (quite hard as a first language), and the more introductory route of learning Microsoft's XNA platform, which allows you to build games for Windows and the Xbox 360 using C#.

Personally, if I were just starting out I would learn XNA, build a game, and try to get a job at a legitimate game house where I could learn C++ on the job since the time investment is huge.

Resources for both approaches are below:

XNA

XNA Game Studio Express: Developing Games for Windows and the Xbox 360

Beginning XNA 2.0 Game Programming

XNA 2.0 Game Programming Recipes: A Problem-Solution Approach

C++

Beginning C ++ Through Game Programming

Introduction to Game Programming with C++

What is the greatest reward from learning code?

The greatest reward for me has been the ability to live anywhere in the country, earn a great living, work for myself, and create new things from nothing.

How can I find a software firm that will train me?

This is a sought after situation. I posted about this a while back in Does Anyone Know of a Real Software Apprenticeship?, and received few (if any) solid leads.

There are internships available, but a true learning apprenticeship where someone will train you to code is tough to find. Much more often you will be assigned a project and handed a programming book for a language you've never used. This may sound daunting, but if you have a head for programming you will learn language syntax very quickly.

I've never heard of apprenticeships being advertised in job postings – most of the time you will stumble into them informally by working on a project with a senior developer who takes the time to teach. So my advice is that when you go in for an interview, try to determine if the company has a teaching culture where this kind of thing might take place.

How difficult would it be for someone who knows computers but is not a computer geek to become a programmer?

I was not a computer geek when I became a professional developer. You don't have to be a geek to start writing code, and you may find that learning to code leads you to other less-geeky things like web design or usability design.

I would probably be considered a computer geek now, though, so be forewarned that it my suck you into the dark side.

Is Computer Science the Same as Programming?

One comment on my post Advice on How to Become a Programmer reads:

> "…Computer Science is NOT programming. That is like saying materials science is construction. Unfortunately, most students aren't aware of this difference until halfway through their degree. If you like fairly abstract math theory, or want to deal with advanced models, do computer science. If you just want to write code, maybe take programming course, but then practice!"

Well put. Computer Science is a discipline dealing with the theory of computer software and algorithms. It's theoretically possible to earn a degree in Computer Science and never write a working line of code.