How to Become a C# Programmer

A systematic approach to learning C# that anyone can follow

by Scott Whigham

LearnItFirst.com founder Scott Whigham talks about how you can become a C# programmer by following a step-by-step approach that requires minimum investment, no college degree, and no prior experience.
Table of Contents

Introduction 4
“The System” 7
Step 1 9
Step 2 13
Step 3 14
Step 4 16
Step 5 20
Step 6 22
Step 7 23
Step 8 25
Step 9 25
Step 10 26
Step 11 28
Step 12 30
Super Secret Mystery Bonus Step 13 31
Final Thoughts 32

Appendix

About the Author 35
Alternatives to Using this System 36
Sample Pet Project Feature List 40
LearnItFirst Coupon 42
Introduction

The idea for this book came about one day when I received two emails from two people in completely different hemispheres of the world, both asking the same thing: “How should I get started trying to become a programmer?” Getting emails like this is nothing new for me – as a trainer/presenter and founder of a training company (LearnItFirst.com), I’ve received hundreds of these types of emails through the years. When I received these two nearly identical emails on the same day, the idea came to me that I should really spend some time and formulize what I had been telling people about in my email responses – and this book is the outcome.

What follows is a twelve step approach to becoming a C# programmer. I’m a big believer in the old axiom, “If you want to be successful, find someone who has had success in that area and copy what they did.” I’ve seen it work for myself and friends time and time again and that’s really the genesis of this system: the principles behind these twelve steps come from reading about successful developers, talking with good and great programmers (there is a big difference between a “good programmer” and a “great programmer”), and from my own experience on the path to becoming a developer.

Who was this book written for?

This book is written for someone who is just getting started with C# or for someone who has limited experience in programming. If you’ve been a programmer for 20 years then you may still get something from the material but you really aren’t the “target audience” for this book. If you are just getting started in your programming career – even if you are an employed programmer – you will definitely get something from it.

Which version of C# does this cover?

Any and all – this book is more about a systematic way to learn how to become a C# programmer than it is learning a specific version of C#.

What this book is

This book outlines a systematic approach to answering the question, “How do I become a C# programmer?” If you are trying to go from “know nothing” or “know a little” to someone who is “highly employable” as a C# developer, then this book is for you. This book does make the assumption that you not only want to become a C#
programmer but that you want to become a good C# programmer. The steps and the work outlined herein will help just about anyone make the leap from newbie to good C# programmer.

**What this book isn’t**

This book will not teach you C# - it isn’t about coding, code samples, syntax, definitions, type systems or anything of the like. You will need multiple other resources to help you actually learn the C# language but don’t worry: this is specifically addressed in several of the steps in this book. This book is more about the “metagame” of becoming a C# programmer; it talks more about all of the little things that you must do in addition to just learning syntax and application design. Most beginning C# books and courses focus on language-specific topics like data types, design patterns, and how to create well-designed C# applications but few discuss anything beyond the syntax. That’s where this book comes in – it will tell you exactly what to look for in a book or course, what the next steps are beyond your first book, and much more.

**Will this system work for you?**

The “system” requires a great deal from you – you must make the right choices along the way, put in the amount of time necessary, and not get sidetracked among other things. So it’s really up to you – are you going to commit the amount of time necessary to complete the system? Will you solemnly swear not to get distracted along the way? It won’t be easy, I promise you that much. It will be rewarding but not easy.

We also have to consider that not all currently-employed C# programmers are “good C# programmers” (after all, you don’t just want to become a C# programmer, do you? You want to become a good C# programmer). There are many, many C# programmers who work day in, day out on mission critical projects for their employers yet they are not “good developers”; some are even downright bad programmers! We can thus categorize the quality of a programmer using a pyramid:

![Programmer Quality Pyramid](image)

There are more “bad” programmers than “average” programmers (though this is often because of the learning curve required to become average; you have to be “bad” before you can become “average” and so on) and there are more “good” programmers than “great” ones. Would you be happy if you followed the system and only became an average C# developer? That might mean a nice job with a great company and a $70,000 USD a year salary. Lots of folks would be happy with that...
Here’s what I believe: the system will work for anyone who has the aptitude to become a good C# programmer. However I also believe that not everyone has the aptitude to become a “good programmer” (whether it is C#, Java, Lisp, or any other language). This is liable to ruffle a few people’s feathers – people don’t want to hear that either they or someone else might not even have the option to become a good programmer. While I don’t want to get into “What is a good programmer and how are they different from an average programmer?” – that could be a separate book altogether – I will say that, in my opinion, good programmers generally have the following qualities:

- Average to above-average intelligence
- Ability to remain focused on accomplishing a task
- Creative
- Ability to break down complex scenarios into more easily digested chunks
- Considered to be good problem solvers
- Enjoy programming

If you have those qualities then, yes; I think that, if you follow this system, you will become a good C# programmer. If you are lacking in those qualities, that doesn’t mean you are destined to “only” be an average programmer. It might mean that you have to work a little harder than someone who was blessed with all of those attributes in abundance. Or maybe you will “just” be average. As I said earlier, being an average C# developer can still be very rewarding.

There’s only one way to find out – follow the system and see if it works for you. And that’s the same reason that sports games, card games, or any other competition is played out to the final minute: you can have a fighter who is a 100-to-1 favorite to win yet he gets knocked down by the underdog. The bottom line is that you won’t know until you try. You might be thinking to yourself, “I have all of those attributes – I’m going to become a great programmer!” and you might be right. Or you could be thinking, “Hmmm – I don’t know... I’ve never done anything like this before. What if it doesn’t work out? I’ve started a lot of things that I’ve never finished before... I don’t think I can do it.” and you might be right. I don’t know. I’ve seen lots of brilliant people try yet fail and I’ve seen normal, everyday people blossom and surprise me. I know enough to know that I can’t tell you who will become “good” or “great” just by looking at a resume or because they went to this or that school.

Try. Don’t give up. You will get discouraged. But if you continue to “work the system” and apply the principles, I have full confidence that you will succeed.

I wish you luck and I hope that you will share this advice with others so that, together, you and I can make someone’s life a little bit better.

Scott Whigham, author
How to Become a C# Programmer

by Scott Whigham

The System

Let me start out by discussing why I choose to use the phrase, “The System”. I didn’t have to call it anything; I could’ve just said, “Hey – go do this stuff and you’ll become a C# programmer.” The reason I chose that phrase is because what you are reading is really a systematic way to go from A to Z. If I had called it a “step by step approach”, it wouldn’t have changed anything.

You should also know that this isn’t really my system; I actually did not come up with a single step on my own. My input was simply to collect the information from interviews and discussions with developers, infused these with a few of my own thoughts and experiences, and then coalesce it all down into a pragmatic approach to get from Point A to Point Z. As I mentioned in the Introduction, the ideas came about from finding what other successful people had done and turning it into a formula that someone new to programming could follow. The System is a 12-Step program with a Super Secret Mystery Bonus 13th step at the end (don’t you dare skip ahead!).

The Steps

Step 1. Pick a Pet Project
Step 2. Buy a starter-level C# book or video course
Step 3. Begin reading the book or taking the course
Step 4. Implement your pet project while reading the book/taking the course
Step 5. Post your questions in an online C# forum
Step 6. Find some C# sample apps and review how the developers did things
Step 7. Finish your book/course and buy another beginner/intermediate level C# book/course
Step 8. Begin answering other people’s questions in your C# forum
Step 9. Pick a new pet project and begin implementing it
Step 10. Review some of the C# open source websites and/or applications
Step 11. Contribute to open source
Step 12. Write a few articles on C# topics you enjoy
Step 13. Super Secret Mystery Bonus Step
You didn’t just skip ahead to Step 13 already, did you? That would be naughty... You want to become a good C# programmer, not a naughty C# programmer.

And while I’ve been around the block long enough to know that any sufficiently advanced developer could look at my list and add/remove/change things, I do think that, if you follow these steps (or even follow a significant portion of the steps), you will be able to start from scratch and go on to become a good C# developer.

**How Long Will It Take?**

Let’s get the bad news out of the way: this isn’t going to happen overnight. It may take you six months or six years – I don’t know. I think that, if they were really focused on “working the system”, most people could finish the steps in six months. However there are some who could finish it in three months simply because they are a “natural”.

Is that disheartening? Did you hope I was going to say “one month”? I’m sorry – being a programmer is hard. You don’t have to work this system – certain universities offer great Software Engineering degrees.

But if you follow the steps below, I believe that you will become what you ask for: a good C# programmer. And C# is a good choice incidentally - C# developers in the US in 2009 made anywhere from $60,000 - $120,000 on average according to http://www.payscale.com/.

**You Do Not Have to Have a College Degree to Become a Good Programmer**

A lot of people think that they need to have a Computer Science degree to be a “good” programmer but I disagree. I don’t have such a degree and I fancy myself a more-than-able developer. Computer Science degrees are wonderful things but they aren’t required to be a programmer or even a good programmer. Will it help you be a better developer? “Probably.” I say that simply because so much depends on what classes you take and how well you understood each. But I don’t think it’s a requirement to be a good or even a great programmer.

Now don’t misunderstand me: I think it’s great to take computer science classes but I don’t think that having a Computer Science degree automatically makes you a “good programmer.” You could, I suppose, make the case that a Software Engineering degree would fill that gap and I might be more inclined to agree.

There is a separate topic that you may want to investigate: “Will not having a CS degree prevent me from getting the job I want?” That’s not a question I can answer for you; you need to spend some serious time on that on your own.

**Do I Have to Be a Math Guru?**

Again I say, “No.” It might help you, in certain situations, to know complex mathematical structures but I’ve made a career out of being a programmer and I failed Trigonometry three times (well, “failed” is perhaps the wrong word - “stopped going to class” or “couldn’t get up in time after playing music in bars all night long the...
night before” might be more appropriate. Seriously – scheduling any class at 8:00AM for a 20 year old college student who pays his bills by playing music is just mean).

I will say that, because I don’t have a mathematical background, I am ineligible for certain jobs. Let’s just end this section with this: if you want to work on applications that require the programmer to have an in-depth knowledge of complex mathematics, then plan on taking (and excelling in) lots of math classes.

### Step 1

**Step 1: Pick a Pet Project**

The first step to becoming a programmer should be exciting and fun since, for so many people, programming is fun and exciting. Learning C# doesn’t have to be a chore; for many people (like me), programming is an absolute joy and I even love learning new languages.

My favorite way to learn a new language is to start with a “pet project”. The idea here is that you find some little project that you want to build and you go and build it. At the end of building it, you have given yourself some practical programming experience and learned quite a bit along the way. It’s not quite “trial by fire” but there’s an old computer programmer axiom: “The compiler doesn’t lie.” Your program either compiles or it doesn’t. You will learn right from wrong the old fashioned way: by screwing up and having to learn how to fix it (a.k.a. debugging).

This step requires no knowledge of the C# language and no additional resources (like books or classes); it relies solely on your creativity and what you think would be (a) fun to implement, and (b) would provide you with enough challenges that you will have a sense of accomplishment upon completion.

“But Scott - what should I choose as my pet project?”

Easy - something “moderately ambitious.” After completing your pet project, you want to have a pretty, well-made application that, if you were asked during a job interview to talk about the architecture and show it to the interviewer, you would feel a sense of pride. In fact, hiring managers will often ask you to bring/share copies of software you have written (or at the very least show example code). Here are a few suggestions to get you started:
I could list dozens more as well but the common theme each of these has is that (a) they are moderately ambitious, and (b) you’ve used something like them before therefore you can focus more on the programming side of things rather than the innovation side.

Here’s another idea: find something you like to do in the non-programming world and write an app that helps you in some related way. I’ll give you a personal example: I was training for a marathon right about the time when .NET came out so my pet project to help me learn .NET was a tracking app to log/report my run history. I could enter my times and distances and view reports on averages, fastest, slowest, farthest, etc. It was fun and it helped me learn what I needed to know at the time (both about .NET and about my running) – and that’s what the pet project is all about!

**Your App Will Suck**

Here’s something you should know early on: *you will do it wrong*. The architecture will be wrong. It will not be fast. It will be buggy if put to the test. And, if you follow all the steps of my system, you’ll look back at this app and laugh at how silly it is a year from now. But you know what? All beginning programmer-level pet projects are that way and, since you have to start somewhere, this is just what you do. Don’t get intimidated by the sheer magnitude of what you need to learn in order for this pet project to become “great”; chances are that, by the time you know all the things it takes to build *great* apps, your mind will be on to much more exciting applications. Just consider it a rite of passage. As Guy Kawasaki, the famous marketing manager of Apple’s early days and current Garage investor, so famously said, “Don’t worry, be crappy.”

**Learning about Scope**

Another thing you should start learning early on is the concept of *scope* and how important deadlines and prioritization are to software engineering. To define the scope of your project, do the following:

1. **Get your app description down to a single, sexy sentence**
   - In the investing world, this is known as your “elevator pitch”. Pretend you are on an elevator with someone who, if they bought and liked your app, could change your life forever. What would you say in a single sentence that tells them what your app is and does?
   - You can see a sample in the example feature list in Appendix C

2. **Get a pen and paper and write down the top 25 features your app should have**
   - Spend some serious thinking time away from the computer here to come up with your features
   - Next, review similar apps and see what you’ve forgotten or where you have added value
   - You can see an example feature list in Appendix C

3. **Pick a date that you’d like to have your application finished by**
You want to give yourself enough time to finish the app but not so much time that you lose interest in the app because you have advanced beyond it.

If you can devote 2-4 hours per day five days a week, it should likely take you 2-4 weeks to complete.

Let’s call this the “project due date”

4. Categorize your feature list
   - Separate the features into three categories:
     - **Critical** - must exist for the app to be considered an [insert type of app] app
     - **Important** - these features generally exist in most of these types of apps
     - **Like to have** - would love to have them but, if they aren’t there, then it’s okay

5. Assign timelines to each task in the Critical category
   - Go through each feature and guesstimate how many hours it will take to fully implement that feature.
   - To “fully implement” a feature requires (1) thinking about it, (2) coding it, and (3) testing and debugging it.
   - No one expects you to do this correctly - after all, you haven’t had any real training. Just do the best you can.

6. Decide what gets included in version 1.0 and what gets cut
   - Add up the timelines for your Critical features - does it exceed the project due date?
   - If your project still requires Critical features yet you have no more time, reassess the accuracy of your project deadline and each item’s individual timeline. If it’s still too much, you need to reassess whether or not you’ve accurately defined what “Critical” means to your app. Having a “great looking app” is not critical but “user can save their work so it will be there next time” is.
   - During this phase you will also notice dependencies between the features (i.e. you can’t implement this feature until or unless that feature is added or completed.

It may seem backwards to pick the project due date and then to pick which features are to be included based on how much time you have. I chose this sequence on purpose simply based on my own past experience. As a novice programmer, I would come up with these massive pet projects with each feature more awesome than the previous. I would eagerly start working on implementing my app and would just get so excited while working on it that I would get carried away.

The problem invariably was that I let myself keep adding features to the app instead of finishing it (this is called “scope creep”). I’d usually end up not finishing it due to distraction (got a new job/wife/girlfriend/etc), disillusionment (“This is just too big – I’ll never finish it so why bother?”), disinterest, or any other number of reasons.

The point is this: **the farther out your pet project’s due date is, the less chance you will finish it.** You need a sense of urgency: “I only have four weeks – I better not slack off tonight.” By having a predefined project due
date and then arranging your activities so that you hit that due date, you are giving yourself the best chance at success.

There is an art to being able to plan the project due date. I think of it like a bell curve:

You can’t finish it too quickly (since you haven’t even started Step 2) and, the longer you put it off, the less chance that you will even want to finish it. So give yourself a difficult to achieve deadline and stick to it.

Tip #1: The farther out you put the pet project’s due date, the less the chance you will finish it

The key lessons from this exercise are that you need to learn how to choose which features will go into your applications, you need to be able to guesstimate how long it takes to finish each, and you need to be able to meet a deadline. Generally speaking, the first version of an application has far fewer features than version 2.0. Most of the time, the main features in version 2.0 were originally thought of prior to shipping version 1.0 but, due to prioritization and time constraints, they got bumped to version 2.0. And don’t worry - you’ll get this wrong too (even seasoned pros get this wrong often). I’m terrible at this personally!

Learn to Ship on Time

One of the most famous and oft-quoted lines regarding developers was penned by Steve Jobs (of Apple fame): “Great artists ship.” In other words, you can draw up the most elegant of applications but if you don’t release it (i.e. ship) then it’s nothing. Have a deadline and work like #### to make that deadline. There may be delays - there are almost always delays in software - but ship quickly. Again: you will mess up and it won’t be pretty. Get over it and just ship it!
Databases Are Everywhere

You’ll notice that, in most of my suggestions, there is persistent data storage required - i.e. each time you execute the app, you want access to the data/state you previously entered. Do not shy away from this! Most applications today, at some level, use persistent data storage (in the form of cookies, XML files, a database, etc.). Step 3 will help you with this...

One Final Suggestion

Although you could make your pet project a web application, I would advise against it for the time being. You’ll have plenty of time later in “The System” to do web apps or any other type of app. For now, I suggest that you just stick to a plain old Windows Forms-based app. You could also make this a “Console” application but I think that it is better to do a GUI-based application at this stage.

Step 2: Buy a starter-level C# book or video course

As the founder of the web’s greatest video training company (ahem – LearnItFirst.com, that is), I’m obviously biased towards video training. In fact, I have what I believe to be a very good C# training class for Visual Studio over at http://www.learnitfirst.com/. However I also love reading books. So pick up a video course or a book – but just pick one for the time being. I’ve found that if I buy too many books when I’m first learning a language that I get overloaded and it makes the process of learning a language more difficult. However, what works for me may not work for you so don’t just do what I say because I said!

What to Look For in a Starter-Level Book or Online Video Class

Obviously you want good coverage of the topics that will help you build your pet project. This likely includes:

- Building GUIs
- Working with persistent data storage
- Working with files and folders
• Basic architecture
• Basic object-oriented architecture and design (OOAD - also referred to as “class design” in certain places)
• Chapters that can be digested individually
  o You don’t want a book/course that, if you skip an early chapter (say, Chapter 5 for example), you can’t finish the rest of the book.

What You Don’t Care About

Many beginner-level books include huge, intricate sample projects that span multiple chapters that are meant to replace Step 2 in my system (the pet project). You’ll recognize such books when you see marketing speak on the front cover such as, “Learn how to build a Password Manager!” or, “Build Your Own Web Server.” I’m not a fan for two reasons:

1. I’ve never once finished one of these long, multi-chapter sample projects despite starting several
2. I think that you will learn faster and retain more by having to learn how to implement your own application without anyone there to “hold your hand” (i.e. by giving you the code files and step-by-step instructions)

Sample projects have their place but I think smaller projects are better at this stage. Again: look for books/courses that have decoupled chapters (i.e. chapters that can be taken independently).

Step 3: Begin reading the book/taking the course

Obvious step - couple of things to note:
Just Because They Are the Expert Doesn’t Mean They Know Everything

Speaking as both an “expert” and an educator, I can tell you the inside truth: experts are experts in what they know. I know that sounds simplistic but it’s true: you can ask me tons of questions about things I know and I will dazzle you (maybe?) with my intricate knowledge of this or that particular piece/function/method/facet/property/etc. But ask me questions about complex topics that I have no real-world experience in and I’ll fumble around like anyone (and I’ll look very little like an expert).

Whenever someone tells you about a “best practice” or you hear phrases like, “That’s the way everyone does it”, it’s a good idea to have a reality check and verify what they are telling you is true. At this stage of your development, you likely won’t have the knowledge to ask smart questions so it’s important to keep notes of topics/areas that you’ll want to cover later in your development. You may find that you will learn things later on in this system that go completely against what you learn in Steps 3 and 4.

You know what? You will also misinterpret many things and completely gloss over incredibly important topics. This is normal.

Keep a Notebook and Code Manager

Speaking of keeping notes, you will have so many questions at the beginning of your adventure to become a better C# programmer that you need a way to organize them. I’m a huge fan of OneNote and I think it cut down the time it takes me to learn a new language (http://office.microsoft.com/en-us/onenote/). If you prefer another note application, use it. Prior to using OneNote, I would create compiled help files (.chm files) so that I could carry them anywhere. Use whatever works for you. I like OneNote because it serves as both a notebook and a code snippet manager but I have lots of .sql files, .cs files, .vb files, etc just randomly stored in folders as well.

In your notebook, you’ll want to have sections/categories similar to:

- Questions I Have
- OOAD
- Data types
- Working with Files and Folders
- Best Practices

- To-do list
- Working with Databases
- MP3 player
- Code Snippets
- Tips & Tricks

In each section/category, you’ll take notes and maybe even reference page numbers from your book or video course, web pages, specific concepts, etc.

Tip #2: Use good SIPs (Statistically Improbable Phrases) in your notes and code samples to make it easy to find topics later.
Tip #3: Include descriptive comments in any code samples you create or download. I can’t tell you how many times I have opened up a folder, seen 50+ .cs files (C# class files), and just been clueless as to which one contained an example of working with x, y, or z method.

Tip #4: A good technique for learning is to take notes on a sheet of paper and then transfer them to OneNote later. This allows you to participate in the learning process three times: (1) when you read/heard the material [i.e. aurally or visually], (2) when you physically wrote it down, and (3) when you physically typed it in.

Step 4: Implement your pet project while reading the book/taking the course

As soon as you find something in the book or course from Step 2 that you can use in your pet project, go for it! However don’t be too impatient; it’s not until you get a few chapters into it that you will have the knowledge of how to even get started.

Here is a basic strategy to help you get started:

1. Start with the GUI
2. Code up the events
3. Write up the pseudocode
4. Do your object-oriented analysis and design (OOAD)
1. Start with the GUI

This probably goes against the grain of much conventional programming advice but, since it’s my “system”, I can tell you what I think works best. I find, when I am learning a brand new language, that if I start with the visual elements of my very first pet project then I can visualize/think of the rest of the application much easier. It gives me a frame of reference, if you will, and allows me to quit thinking in completely abstract terms and lets me hone in on accomplishing specific things (like what happens when a user clicks the Submit button). Later on in your career you will learn of different techniques but, for now, this is an easy way to dive in.

Don’t get too fancy here - start with the first screen that the user will see; it is okay to add fancy items like splash screens and product icons if you wish. You’ll be doing GUI drag-and-drop at this stage of your career. You can’t screw anything significant up at this stage and you can’t do it wrong - if you make a mistake, just delete it and drag-and-drop another item onto the surface. The key thing here is not to get so bogged down in the architecture that you have to re-do too much work later on (called refactoring).

How to Design a GUI

While everyone has their own way of doing this, let me offer a suggested system for designing your first GUI:

1. Start with a pen and paper and write up what it should look like (i.e. where the icons and buttons go)
2. Continuing with pen and paper, write up what menu items should be (i.e. File, Edit, Help)
3. Once finalized, just use the Visual Studio to mock it up. Visual Studio has such great drag-and-drop graphical features that it makes it easy to rearrange and do layouts. Later on in your career you may opt for more sophisticated tools but, for now, keep it simple.

Tip #5: It’s okay to skip ahead/around in your book/course (from Step 3) to the section that talks about how to create GUI elements!

Once again, this is just my style and that doesn’t make it the “right” way. However, if you haven’t had any practice designing applications before, it’s often easiest to start with one technique that you know works and then develop your own style after you’ve had more experience and/or tried multiple techniques.

After completing the GUI design, you will have a pretty application (if you did the graphic design well) that doesn’t yet let users do anything.

After the GUI, Start with Events

The next thing you want to do is start “wiring up” the buttons and menu items. This is where your programming skills will begin to develop. The actual work is done when you decide the application’s response to things like:
• What the app does when something loads or unloads (i.e. appears on the screen and leaves the screen)
• What the app does when a user clicks a button or menu item
• What the app does when the user clicks Maximize, Minimize, or closes the application

As a C# developer, you’ll spend much if not most of your time determining what happens when an event occurs. Events, as your book/class from Step 2 will teach you, are the application’s response to something occurring (like what happens when a user clicks the “Submit” button). Go ahead and get your events in place - you can add the actual functionality later on.

“But Scott – how do I do this?”
This is really something you should learn from your class/book from Step 2. In the Visual Studio, if you double-click on a button, it will automatically generate the OnClick() event for you. Just look for “events” in your book or class for help.

“But Scott…. What should my events do?”
Nothing yet; just get them in place and we’ll cover the work a bit later.

After completing this step, your application will respond to events (i.e. clicks) but won’t perform any actual work.

After the Events, Do Your Class Design

The reason I advocate pushing this step until after you have both the main GUI elements and main events in place is that, if you start with the class design and then do the GUI + events, inevitably you find how screwed up your class design was and you have to rip it out and start over! It will happen even by delaying the OOAD until this point but the chance is that, by delaying this step until you have semi-finalized your design and events, it will happen to a lesser degree than if you had just started with OOAD. Later on in your career you may opt to start with OOAD/class design but, for now, it’s just easier to start with the GUI, then determine the events you need, and finally to determine the OOAD. Again, this is my suggestion but it doesn’t mean that it’s right. If your book/course/friend suggests another technique that you prefer, go for it.

After completing this step, your application will still respond to events (i.e. clicks) but won’t perform any work since you’ve only designed the classes, not implemented them.

After the Class Design, Implement Method Stubs

Method stubs are a way for you to “kind of” get your program working. When you create in this manner, you will go ahead and create your objects along with the methods and properties for each. What you won’t do, however, is to finalize the method/property code; you’ll put a stub in place instead. Think of a method stub as a placeholder for work to be done in the future. In most cases, you won’t have the necessary knowledge to write each method/property yet so go ahead and create the stub as a placeholder.
Tip #6: Use exceptions to help you remember what is a method stub and what has been implemented

Tip #7: Use comments in your code to tell Visual Studio that this is a TODO list item:

```csharp
void GetToDoListItem (int itemId)
{
    // TODO: This is a method stub - finish it!
    throw new NotImplementedException("Finish me!");
}
```

After the Method Stubs, Write the Pseudocode

Learning to write pseudocode is a critical skill. Pseudocode is how you write the “how” of what your method/property will do but you write it in a programming language-agnostic format. I’ve seen very complex pseudocode examples and I’ve seen bullet points on a piece of paper serve the same function. To me, it doesn’t matter - use whatever helps you understand the goal of the method/property. I’ve found many times that, in writing the pseudocode, that my logic was completely backwards and that I needed to rethink how the application worked. That’s awesome because it means that I didn’t have to spend a ton of time refactoring the application!

Your book or class from Step 2 should have plenty of information on writing pseudocode. If not, a quick peek at Wikipedia should help: [http://en.wikipedia.org/wiki/Pseudocode](http://en.wikipedia.org/wiki/Pseudocode)

Once you have the pseudocode written, get to work finishing the app!

---

**Step 5**

Step 5: Post your questions in an online C# forum
While you go through the Steps 1-4, you will likely:

- Get excited about the possibilities that C# has to offer
- Become dazed and confused
- Get utterly frustrated with how vast C# and the .NET Framework are
- Get utterly frustrated with how complex software engineering is
- Mess up royally by either overwriting changes that took you hours to complete, forget to backup your files, or need a complete refactor

You know what? It’s all normal for you at this stage. Your life as a C# programmer will likely follow the Gartner Hype Cycle:

The “Technology Trigger” is your deciding to learn C# or to become a good C# programmer and the “Peak of Inflated Expectations” is what happens when you start seeing C# as the solution to everything. The “Trough of Disillusionment” is when you realize that (a) C# isn’t the solution to everything, (b) that C# is massive, (c) that writing software is hard, (d) that there is so much to learn, and (e) that this isn’t going to be as easy as you thought. The “Slope of Enlightenment” is when you start to “get it” and the “Plateau of Productivity” is when you finally are able to get some quality work done!

Trust me - it’s going to take a lot of time, a lot of practice, and a lot of banging your head on your desk but it will be worth it! I love programming – it is such a rewarding experience – and I’m glad I put in my time even after all these years.
Your Goal: To Minimize Your Time in the “Trough of Disillusionment”

It absolutely sucks to be bitterly disappointed with what you’ve chosen to do. “This is way too hard for me - there’s just so much to learn that I’ll never learn it all!” While you may not be able to “learn it all”, you can learn enough, I promise you.

There are several techniques to minimize your time in the Trough of Disillusionment, a few of which are listed here:

- Read/study more (i.e. the brute force technique)
- Find a mentor
- Ask for help
- Sleep on it/take a break

I find that putting a problem aside for a day or two often refreshes me and allows me more perspective - don’t be shy about knocking off and going to the gym or whatever it is that floats your boat. I remember when I was developing LearnItFirst - I would work these 70-80 hour weeks and then hit a wall of disillusionment (“There’s no way one person can do all of this!”). The only thing to do was to just take a break for a few days - we would go on mini-vacations or I would take my kids to a play center. It worked wonders for me.

If you have a friend to ask for help, then your life is made just a bit easier. Most newbies aren’t so lucky - they may not know any programmers or, if they do, the programmers may be at their job and if the newbie lets on how much they don’t know, they could get in trouble at work. That leaves the internet!

Find an Online Forum and Ask Questions

We “good programmers” expect you to do this and, if you don’t (or don’t follow forum etiquette), we think far less of you. It shouldn’t be difficult to find a friendly online C# forum –just use your search engine of choice. And I can’t say this enough: learn how to post! Learn how to ask questions by reviewing how others have asked questions. There are several good articles on netiquette:

  - Which features my favorite topic: “Don’t hijack the thread!”

You Will Get Stuck in Your Pet Project

Ask questions in the forum so that you get out of the Trough of Disillusionment and on the path to the Slope of Enlightenment!
Step 6: Find some C# sample apps and review how the developers did things

Chances are that, while completing your class/book from Step 2 and your pet project, you will come across at least one sizable sample project written in your chosen language. While it is always fun to peek at other’s code, let me offer a bit of advice:

1. **Look for complete applications**
   - In my experience, complete applications tend to be better vetted, contain better code, and have less errors than just quick “How to” example code.

2. **Check out the author(s)**
   - Authors often contribute on multiple projects so, when you find a good contributor/author, check out what else they have that you can download! You also might wish to see if the author/contributor has a blog, a book, a class, etc.

For the Microsoft community, [CodePlex.com](http://www.codeplex.com) is generally the hosting site of choice for “sample” apps and general release apps alike. For many applications, you can download both a compiled version and the source (often in both C# and Visual Basic). There are literally hundreds of high quality sample applications to choose from - here are a few that have C# source code included:

<table>
<thead>
<tr>
<th><a href="http://www.codeplex.com/">http://www.codeplex.com/</a></th>
<th>CodePlex.com has literally thousands of apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Studio samples</td>
<td>Installation options</td>
</tr>
</tbody>
</table>

Don’t look for full-blown or overly complex apps just yet – you might find that you get lost. Keep it simple.

**Be Wary**

Just because the code works and has pretty graphics doesn’t mean that it is well-written code...
What to Look For

You’re really wanting to see the big picture - the “how” of why the authors/contributors did it the way they did. You aren’t so much looking for specific code examples as much as you are looking to see how professionals ship code. Look for things like:

- How did they create the solution: one project or multiple projects?
  - If multiple projects, why?

- How did they handle class design?
  - It’s often handy to pick a problem and try to decide how you would’ve designed the OOAD and then compare it to how the authors did it.

- Do you get any compiler bugs?
  - Can you fix them?

- Did they do things the way that your book/class author (from Step 3) showed you?
  - What is different?

Don’t be surprised if you find it difficult if not impossible to answer all of the above questions - after all, the people who contributed to the sample app you just downloaded probably have several more years of experience than you do. But that’s okay – do your best. You’re well on your way to becoming a good C# programmer!

Step 7: Finish your book/course and buy another beginner/intermediate level C# book/course

You have two choices for continuing your education after you’ve completed your pet project and first C# book/class:
1. Continue with domain-specific knowledge (i.e. get another C# book/course)
2. Branch out to software engineering topics like OOAD or design patterns

Personally, at this stage, I think you’re better off going with option #1: getting a C#-specific book/course. I know there are tons of folks who would disagree - and maybe you should listen to them! - but I think that, at this point in your career, it is quite difficult to wrap your head around abstract concepts for months at a time; instead I’ve found that people learn better when they can apply what they are learning to their language of choice. There are plenty of C#-based software engineering books and courses out there that combine the two.

However, I would suggest you pick a next-level book/course that helps you acquire specific skills that you can use to get a job. Learning design patterns in C# is great but they won’t open files, make web service calls, do type conversion, or teach you boxing/unboxing. Right now you should be 1-2 months into your C# career and, in my opinion, you still need to learn more of the basics of C# before getting into topics like design patterns and OOAD specific topics.

So focus on finding a beginner/intermediate level book that helps you accomplish something that you can get paid to do:

- ASP.NET (now’s a good time to start with web apps incidentally)
- How to work with SQL databases
- Working with Silverlight
- Pick up a comprehensive course/book on the .NET Framework

I’m sure that this advice is just “crazy” to some people – you’ll find that book/career advice in the programming community has lead to many a flame war – but I think this is the best approach. Take my advice with a grain of salt. Does it make sense to you? Then I hope it helps. If it doesn’t make sense, don’t do it!

**What to Look For**

Since Step 9 is coming up and it is all about picking another pet project, pick a book/course that can help you develop your next pet project.
Step 8: Begin answering other people’s questions in your C# forum

Now that you know a little, you can start helping others. One of the most proven ways to learn is by teaching others. You may think you know how things work but trying to explain a tough topic to someone who knows less than you when you don’t fully understand the topic is tough for anyone! So help others - look for the newbies and try to guide them along. Make suggestions that reference books/courses/authors/trainers - people love that stuff.

Just remember to follow good forum etiquette. All too often I see a relatively new person getting snippy, snarky, or even downright rude with someone who has just started programming. Remember: you were there too at one point (i.e. you were a noob once yourself). And don’t take it personally when someone is rude to you – either ignore them, report them, or point them in the direction of a few netiquette-based articles. Arguing with people in an online forum is not on the path to happiness.

Step 9: Pick a new pet project and begin implementing it

Now’s the time to move into ASP.NET, Silverlight, WPF, or whatever technology you want to work on. This should be an application that you take with you on job interviews and that, if it’s good enough, you release as
open source software yourself. Take your time, find something interesting to work on, and make sure that it highlights things you like. You’d be surprised at how often employers find job candidates in forums and in the open source community.

Learn About Usability

Software developers are not renowned for creating great, usable applications by default; we have to work really hard to make our applications usable by people who didn’t write the code. Two book suggestions (use your search engine of choice to find these):

- Steve Krug’s seminal work, *Don’t Make Me Think*
- Janice Redish’s *Letting Go of the Words*

Suggestions for Your Final Pet Project

- Take it seriously
- Use high quality graphics (spend a few bucks to get them a la carte from a place like [http://istockphoto.com/](http://istockphoto.com/))
- Embed your name on each source code page
- Have friends test it out

Final Thoughts on Your Final Pet Project

There is nothing wrong with going back to your first pet project and implementing more features...

Step 10: Review some of the C# open source websites and/or applications
This is similar to Step 6 in which you reviewed several sample applications. The difference here is that you want to go with a little more “upscale” applications - more fully implemented apps and more apps written by a community. Here are a few suggestions:

**Various apps:**

<table>
<thead>
<tr>
<th>URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.codeplex.com/">http://www.codeplex.com/</a></td>
<td>CodePlex.com has literally thousands of apps</td>
</tr>
<tr>
<td><a href="http://tweety.codeplex.com/">http://tweety.codeplex.com/</a></td>
<td>Tweety - a C# Twitter client</td>
</tr>
<tr>
<td><a href="http://imagemaster.codeplex.com/">http://imagemaster.codeplex.com/</a></td>
<td>ImageMaster (for managing .iso files)</td>
</tr>
<tr>
<td><a href="http://youdown.codeplex.com/">http://youdown.codeplex.com/</a></td>
<td>YouDown - download YouTube videos</td>
</tr>
<tr>
<td><a href="http://indihiang.codeplex.com/">http://indihiang.codeplex.com/</a></td>
<td>IIS Log Analyzer</td>
</tr>
<tr>
<td><a href="http://tweetcraft.codeplex.com/">http://tweetcraft.codeplex.com/</a></td>
<td>TweetCraft - an in-game Twitter client for WoW</td>
</tr>
<tr>
<td><a href="http://skypefx.codeplex.com/">http://skypefx.codeplex.com/</a></td>
<td>Skype Voice Changer</td>
</tr>
<tr>
<td><a href="http://terrarium2.codeplex.com/">http://terrarium2.codeplex.com/</a></td>
<td>.NET Terrarium</td>
</tr>
</tbody>
</table>

**Open source ASP.NET:**

<table>
<thead>
<tr>
<th>URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.asp.net/community/projects/">http://www.asp.net/community/projects/</a></td>
<td>ASP.NET Starter Kits are a great resource for this stage</td>
</tr>
<tr>
<td><a href="http://www.yetanotherforum.net/">http://www.yetanotherforum.net/</a></td>
<td>YetAnotherForum is a quality ASP.NET forum site</td>
</tr>
<tr>
<td><a href="http://cartrackr.codeplex.com/">http://cartrackr.codeplex.com/</a></td>
<td>Car Trackr is an ASP.NET MVC app</td>
</tr>
<tr>
<td><a href="http://www.codeplex.com/Kigg">http://www.codeplex.com/Kigg</a></td>
<td>ASP.NET digg copy; uses Enterprise Library and LINQ to SQL</td>
</tr>
<tr>
<td><a href="http://www.codeplex.com/oxite">http://www.codeplex.com/oxite</a></td>
<td>Oxite is an ASP.NET MVC CMS</td>
</tr>
<tr>
<td><a href="http://bugnetproject.com/">http://bugnetproject.com/</a></td>
<td>Bugnet is an open source, ASP.NET bug tracking system</td>
</tr>
</tbody>
</table>

**Open Source C# Windows Forms Apps:**

<table>
<thead>
<tr>
<th>URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.nunit.org/">http://www.nunit.org/</a></td>
<td>NUnit is essential for all Windows-based developers</td>
</tr>
<tr>
<td><a href="http://sharpdevelop.codeplex.com/">http://sharpdevelop.codeplex.com/</a></td>
<td>SharpDevelop (a.k.a. #develop) is a free, open source IDE</td>
</tr>
<tr>
<td><a href="http://code.google.com/p/iphonebrowser/">http://code.google.com/p/iphonebrowser/</a></td>
<td>iPhone developer</td>
</tr>
<tr>
<td><a href="http://xmlexplorer.codeplex.com/">http://xmlexplorer.codeplex.com/</a></td>
<td>XML Explorer</td>
</tr>
<tr>
<td><a href="http://turtle.codeplex.com/">http://turtle.codeplex.com/</a></td>
<td>turtle SQL helps you generate SQL Server database docs</td>
</tr>
</tbody>
</table>
The two “biggies” for open source for the .NET Framework tend to be [http://www.codeplex.com/](http://www.codeplex.com/) and [http://sourceforge.net](http://sourceforge.net). CodePlex, because it is run by Microsoft, is a more tightly focused (and possibly more targeted) group of .NET apps whereas I find SourceForge to be a difficult site to navigate.

Note: all links in this book are also online at [http://learnitfirst.com/go.aspx?id=D7FU](http://learnitfirst.com/go.aspx?id=D7FU)

### Step 11: Contribute to open source

You can’t just be a leech (i.e. someone who takes but provides nothing in return) - now it’s your turn to help out. Contributing to open source applications has many benefits:

- You’ll learn to integrate with a pre-existing team
- Your knowledge and comfort levels will be expanded
- You’ll learn the style of others (everyone has their own coding style)
- You will be exposed to technologies/techniques that you would not have otherwise seen
- You’ll feel good that you helped make a product better in some way
- You may receive a level of notoriety by being on the team
- Many companies will not hire people who have not contributed to open source projects
- Being a developer on open source projects is often seen as showing passion for development

Here are a few good resources as well:

- How to start contributing to open source projects
- How I got a job at Fog Creek software
  - [http://hicks-wright.net/blog/how-to-get-a-job-at-fog-creek-part-1/](http://hicks-wright.net/blog/how-to-get-a-job-at-fog-creek-part-1/)

---

© 2009 LearnItFirst.com, LLC

http://www.learnitfirst.com/
What to Look For in an Open Source Project

You want an active community and by “active” I mean that you want to see a steady history of changes, new features, modifications and so forth. Seeing a road map is nice too.

Also, you want something that you care about. Look - we’re all volunteers in the open source world. If you don’t care about bug tracking then don’t join a bug tracking application - you’ll end up quitting or getting booted out because you aren’t spending the time. Also, that road map I mentioned earlier is important - if you see that the application is due to take a turn into a technology that you don’t care to learn, then don’t join.

I also would like to see that the tasks needed to be performed would interest me. If the task list is, “Provide documentation for all classes” well, I might skip that one... Sure it needs to be done but I’m not likely to learn as much performing that task as I am performing a task like, “FTP download routine has a bug - find it and fix it”.

What Not to Look For

Don’t look for polish everywhere - that’s not that important. I’ve worked on YetAnotherForum (http://www.yetanotherforum.net/) for years (and continue to, just to a lesser degree today). When I started on it, it had just changed owners and was going through a lot of changes. It wasn’t well documented, the SQL Server database was poorly designed as were the objects in it, the class design was weak, and so forth - but I learned a great deal working on it. The guys on the project are great guys and they learned as they went too. If I was looking for polish then I would’ve looked elsewhere. However, what I saw was that this software did what I wanted, that I felt I could improve upon it, and that it was in an area that I had a passion for. So don’t get hung up on logos, prettiness of buttons, great descriptions, full-blown documentation - look for places where you can add value. And by the way, today YetAnotherForum is awesome due to the contributions of many members - I highly recommend it if you want an ASP.NET forum.
Step 12: Write a few articles on C# topics you enjoy

Remember back in Step 8 when I suggested that you start answering other people’s forum questions because it helps you learn? Well this is similar: writing an article about a difficult topic will force you to spend a good deal of time on areas that you thought you knew but, upon delving into them, you realize that you’ve only scratched the surface. Pick topics that you like - don’t worry if the topics have already been covered by others; your goal is to help yourself learn to become a good C# programmer, right? So write articles that force you to learn and that force you to simplify complex topics.

“Any fool can make things bigger, more complex, and more violent. It takes a touch of genius - and a lot of courage - to move in the opposite direction.” - Albert Einstein

You want to do one thing: make complex topics seem simple to the reader. Why? Because if you can do that, you have shown you have a deep understanding of the topic. Another great quote is from one of my musical heroes:

“Making the simple complex is commonplace; making the complicated simple, awesomely simple, that’s creativity.” - Charles Mingus

Other Reasons to Write Articles

At this point in your career, you likely either have a decent programming job or are eligible to get one. If you want to move up the career ladder as quickly as possible, you need great communication skills. Just like being a good C# programmer takes practice, being a good author (or trainer) takes practice. Writing comes naturally to some but is difficult for others. Here’s the truth: developers who have “the knack” for creating readable documentation, succinct emails, and who can empathize with their audience will generally move up the career ladder.
faster than those who can’t. By forcing yourself to write articles, you’ll not only learn things about C# and pro-
gramming but you’ll learn how to communicate complex topics quickly and succinctly - and that will pay divi-
dends over your entire career.

Writing articles also gives you a public persona which can be a positive thing (provided that your articles are
quality). Potential employers will do a quick Google search (ahem - or a Bing search) to see what you’ve written,
what you’ve posted online, and generally to see what they can find out about you. Make it easy: provide them
with links to your articles in your resume. I’ve seen people who go so far as to put their forum user names in
their resumes so that potential employers can see their participation!

Where to Publish Your Articles

I suppose “articles” is a bit of an officious word - these could be blog posts if you wish. You have several options:

- You can create your own blog and host the articles there (employers *love* this)
- You can find popular blog aggregation sites and post them there
- You might even be able to post them here (just ask us over at LearnItFirst)
- You could publish them in printed magazines.

Personally I favor creating my own blog and posting things there. I’ve always been that way although I’m not
sure it’s the best thing for you. Check out some of the popular blog sites and see what it takes to have your blog
hosted by them. You’ll give up certain publishing rights yet you’ll be given an audience (and that’s important).

Step 13: Super Secret Mystery Bonus Step

The final bit of advice I can offer is to start your own business! Whether you want to sell commercial software,
have an advertiser-supported ASP.NET website, or do consulting, having your own business is a great way to get
better. It forces you to listen to your users - if you don’t, they won’t pay you. And you don’t have to start full-
time; most developers have side projects in addition to their day job.
If you’ve followed the previous twelve steps, you’ve got enough experience to design apps and to help others. You likely don’t have the experience to be considered a “great” programmer but hopefully that will come with time for you! One of the great things about programming is also true of graphic design: for certain jobs, it doesn’t matter how much experience you have; what matters is that you can get what the customer wants done in a reasonable time frame and at the price they want to pay. If you have a “portfolio” of applications that you’ve written, articles you’ve written, and open source applications that you have contributed to, you can go to many companies and provide great value as a consultant. You also might have a great idea for an application – code that baby up and sell it! It’s quite easy to get software into the marketplace today.

I will say that, if you do decide to hang out your shingle and start your own business, to be successful you will likely need to learn a great deal about other topics such as sales, marketing, and finance to name a few. Don’t let that discourage you – I’m a programmer by nature and I’ve learned “enough” about each to get by. I’m no expert at any of those but it isn’t terribly difficult to master the basics of each. You might not be a great marketing person or a great salesperson but you might find that you are good enough to be happy.

Final Thoughts

I believe that, if you follow the steps that I outlined in this article, you will become a good C# programmer. You just can’t help but become a good programmer when you learn from people with tons of experience (i.e. by reading the right books, taking the right video classes, and reviewing quality developer’s code). It will take you time to become a good C# developer but, like all things worth doing, you will have a sense of pride once you’ve gone through these steps. And if you do Step 13 - the one where you create your own business - you might even be one of those people who makes their living from writing their own software (and that’s a great feeling).

The Next Steps

Inevitably everyone wants to know, “What now? I’ve followed the system and I’m doing well - where do I go from here?” Really it’s just more of a “Rinse and Repeat” mantra: find some books/classes, contribute to more open source projects, develop more pet projects, read more books, take more classes, etc. You need time and “seasoning” to go from “good” to “great”. The good news is that even good programmers can write great pro-
grams, enjoy programming, and make a great living. In fact, it’s quite easy to just be a “good programmer” for your whole life. I hope you’ll go on to become a “great programmer” though!

**What Does It Take to Become a Great C# Programmer?**

Whew - big question! It certainly takes years of experience - I think everyone can agree on that point. What else though? I think most people would list at least some of these attributes:

- High intelligence
- Training on the right things
- Exposure to many systems and styles
- Passion for learning, developing
- Creativity
- In-depth knowledge of internals
- Ability to make complex topics simple
- Unit testing knowledge
- Knowledge of patterns
- Ability to quickly solve problems
- Ability to accurately scope projects and provide accurate timelines
- Ability to hit deadlines
- Ability to debug quickly and efficiently

There are many more attributes and skills, of course, but these are definitely in the ballpark.

**How can you develop these skills?**

I wish I could say, “All you have to do is this one thing - if you’ll go do this one thing, you’ll become a great C# developer” but I just can’t; it’s not possible. In my opinion, some people aren’t born to be great programmers. Some people have all of the attributes except the ability to stay focused and hit a deadline (or lack another critical attribute) - and as such they aren’t great programmers. I do know one thing: you can’t skip the “good programmer” step on the way to “great programmer”. The progression that I listed in the introduction says that you’ll go from bad, to average, to good, and hopefully on to great. So for now, just focus on being good. I suspect that, by the time you’ve reached “good” standing, you’ll know what it takes to get to great.

I’ll give you a few books that will allow you to help yourself (look them up in your search engine of choice):

- *The Pragmatic Programmer* - a must for anyone serious about becoming a great developer
- *Code Complete* - techniques, design patterns, “life as a programmer”
- *Head First Design Patterns* - I think most developers have some variation of this book
- *The Mythical Man Month: Essays on Software Engineering* - You just thought you knew programming...

There are many, many more books worth reading - I’ve only focused on pure programming-related books. There are books that change your outlook on life, books that motivate you, and inspirational biographies that can make you feel like a small, insignificant ant - all of which can help you on your journey. The one thing I’m sure of is that it takes a desire to learn and improve. Without that, the rest will not follow.
One final thing: if you found value in this book, please pass it on to a friend or colleague who may need some help. This book is free (thanks to LearnItFirst.com) and does not require registration of any kind. Feel free to send them the PDF version or to send them over to http://www.learnitfirst.com/books/ where they can download this book (and others like it).

Scott Whigham, author
Appendix A: About the Author

Scott Whigham is the founder of LearnItFirst.com, one of the web’s most extensive video training libraries focusing on technical training. Scott has recorded more than 1,200 training videos on topics like SQL Server, .NET, and more. You can read more from and about Scott at:

http://www.learnitfirst.com/About/Scott-Whigham/Instructor/3.aspx

http://www.techurbia.com/scott-whigham.html

http://twitter.com/ScottWhigham
Appendix B: Alternatives to “The System”

There are plenty of alternatives to this system although not all are available to everyone. To be fair, my system is certainly not the best way for all people (although it might be the best for you); some people may find that learning in a university environment works best for them, for example. Let’s take a look at some of the alternatives:

- Getting a degree
- Get a job and get trained by the company
- On the job training
- Figure it out for yourself

I’m sure that I’ve left a few off but I think this covers the majority of how people acquire programming skills. The main idea here is that you can either have someone else pay for your training or you can pay for it.

Before we take a look at each of these alternatives, let’s talk about how much training costs a company and let’s talk about what you bring to the table if you are already trained. This may seem like an odd diversion but, since three of the four bullet points above involve the company paying for training you and the final one is markedly different – one in which you paid for your own training thus saving the company such cost – I think it makes sense.

The Cost of Employee Training

If a company agrees to train you while you are still considered a “beginner”, they are basically saying, “We will invest many thousands of dollars in you.” This training and investment come occur in one of several ways (or some combination):

1. The company has a coworker train you (or a mentoring program)
2. The company pays for you to attend training classes
3. The company pays for you to buy books or watch training videos

If the company is saddling you on top of a coworker, don’t forget the expense to the company: the coworker’s productivity is now at least 50% due to having to train you (and your productivity is likely 0%). If the training process takes one full month before the coworker can go back to 100% productivity, the company paid the coworker’s full monthly salary yet immediately received only 50% of his or her normal output. And believe me: it won’t “just” take one month; you won’t be at peak efficiency for quite a while and the coworker is likely still responsible for you to a certain degree. In reality it might be 3, 4, or even 6 months before the coworker is back at 100% efficiency.
If they agree to send you to a class, they are paying anywhere from $1,000 - $5,000 just for your attendance. If you need travel costs included, then add in another $1,000 - $3,000. If you take a four-day class, that is also four days of lost work; in other words, the company paid your salary for four full days yet received no direct work as a result. Generally speaking, the cost to the company to send a single employee out of town to a training course is somewhere between $5,000 and $10,000. Ouch!

If they decide to let you go the “self help” route (i.e. training videos and/or books), then their upfront cost is seemingly lower ($100 - $2,000 perhaps) however they are taking a risk: not all people will learn effectively using this method. This approach may also take longer since the training is less job-specific than a training class or working with a coworker may be.

Regardless of which method(s) used, let’s take a look at the outcome: you are a trained programmer with “x” months of experience. The company has invested “y” dollars in you in the expectation that they will earn that money back over time due to your increased productivity.

- You: happy that you are trained and have a career path
- The Company: happy that they have a happy, trained programmer

It’s a win/win situation – that is until the day that you realize that you are a trained and experienced programmer who earns the same amount as a beginning, non-trained, and non-experienced programmer. Uh-oh...

There are many risks associated with training employees and you’ve just experienced the first: if the company hired you as a novice and was paying you $60,000 USD, now that you have received training and have experience you are worth more in the work force than “just” $60,000. What if you decide to put your resume out with recruiters or start looking for a new job? You might be able to get $65,000 or as high as $75,000.

To allay the risk of your leaving, companies will generally ask that you sign a contract of some type that says that, if you leave within a certain time frame, you will pay back the company on a pro-rated basis. These agreements are generally in effect for one year after the training but be careful: I’ve seen companies put a two year clause in and that, in my opinion, is just very damaging to you, the programmer.

The bottom line is that a company has to weigh the cost of keeping you (i.e. the cost of raises, promotions, and training) versus the cost of replacing you. There are companies who try to keep you by offering you raises and promotions as well as companies who try to keep you by forcing you to sign lengthy contracts requiring payback for training (as mentioned). I’ve seen companies hold back on raises for employees who attended training while saying, “We’ll invest this $5,000 in this class for you in lieu of a raise. And, of yeah – sign this payback contract too.” I’ve seen just as many companies (if not more) offer you a generous raise at the six month anniversary because you went through training and stayed with the company.

**The Alternatives to “The System”**

Let’s take a look at each of the alternatives that I listed at the beginning of this section:
1. Getting a Computer Science or Software Engineering degree from a college or university
This is a great idea for most people – college can be such a great experience so why not use that time to get a valuable degree?

- Pros: possibly a great investment, potential networking benefits, few gaps in learning
- Cons: time invested, cost, often do not teach popular or newer languages

Most collegiate programs take four years which is a long time to be either unemployed or under employed. Another challenge is that many universities do not teach classes on modern languages like C#, Arc, Lisp, Ruby or other popular languages and instead they focus on Java or C++ with maybe an “overview” type of class on popular languages. This was more true 5-10 years ago than it is today (2009 as of this writing) so it may be that your university is more sensitive to this and can give you a more attractive curriculum. If you did only learn C++ in college, that makes you less attractive to employers when you get into the job market; the company that hires you does so expecting that they will have to train you on C#, Ruby, etc (which we already know is expensive).

The difference between this and numbers two and three below is that you are paying for a certain percentage of your own training. As such, you can expect to earn a slightly higher starting wage due to the fact that you will require less training (thus you will cost less) than someone in numbers two and three below.

2. Get a job and get trained by the company
This generally isn’t available to non-programmers as a “new hire”; this more likely happens during a time of either transition for a company (such as downsizing or mergers) or at the request of the employee or manager. After all, there are not many companies that will hire someone for a programming position who (a) has no programming experience but (b) wants to become a programmer. I’m sure it happens but not often.

An example of this might be if you were a systems admin at ACME Corporation and, for whatever reason, you wished to become a programmer. If the management agrees, they will let you switch positions and agree to train you. This happens quite often because no company wants to lose good employees. If you are a quality employee, the company will invest in you so that they can keep you.

- Pros: obviously that you are having someone else pay; you may get to take expensive training classes
- Cons: not many; possibly that you “owe” the company; you may be starting over at the bottom of the corporate ladder with respect to seniority; possibly have little choice of what you get to work on initially

In this case we’re talking about someone who had no training and no experience prior to the job and the company pays for 100% of the employee’s training. As such, you can expect a lower starting salary and possibly “lighter” raises initially until you can prove yourself as a valuable member of the team.
3. On the job training
This is another one that generally isn’t available to non-programmers as a “new hire”; this happens most often in the exact same manner as #2 above. The idea here is that you have a company who transitions you into programming but gives you no training dollars – you have to learn on the job as best you can. Good luck.

- Pros: you get lots of hands-on experience; lots of troubleshooting experience; often get to be the sole architect/author of interesting applications
- Cons: gaps in learning; often working on proprietary systems that do not translate to desirable skills by other employers; possibly have little choice of what you get to work on initially

In this case we’re talking about someone who had no training and no experience prior to the job and the company pays for 100% of the employee’s training. As such, you can expect a lower starting salary and possibly “lighter” raises initially until you can prove yourself as a valuable member of the team. This may be doubly true here since we’re talking about a company that tells you upfront: “We aren’t sending you to any of those expensive training classes!”

4. Figure it out for yourself
This is likely a modified version of The System whether intentional or not. In fact, I think it is debatable whether you could even be considered a programmer unless you followed some of the early steps.

This is how I started – I did it because it looked fun, not because I was expecting to become a developer for a living (I wasn’t). I wanted to build some little app or other and, at the time, I couldn’t find one. Someone suggested I write on using C++. I had never taken a programming class in my life but I went out and bought a beginning C++ book and slogged through it until I could write my app. After that, I was hooked and it just went on from there.

- Pros: low cost; choice of what to work on; can be done for fun;
- Cons: no hands-on experience; gaps in learning; time to learn may be longer; less networking opportunities

The beauty of this system is that you can do it part-time, you can do it in lieu of college/university, you can do it regardless of age, and you can do it just to have fun!

And, from a prospective employer’s point of view, you are likely going to require less training than numbers two and three above but more training than if you had a Software Engineering degree. You can expect to be paid close to what you would receive if you had a CS or SE degree.
Appendix C: Sample Pet Project Feature List

The pet project should have a detailed breakdown of exactly what you plan to implement prior to beginning the coding phase. Here’s an example feature list for a pet project titled, “RunReportr”.

One sentence description: RunReportr allows runners to maintain a log of running progress and improvement as well as publish such information on popular social networks.

Feature list:

<table>
<thead>
<tr>
<th>ID</th>
<th>Feature</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Allows multiple users and login</td>
<td>Critical</td>
</tr>
<tr>
<td>2</td>
<td>Data is stored in a SQL Server database for later retrieval and reporting</td>
<td>Critical</td>
</tr>
<tr>
<td>3</td>
<td>User can store data for run date, run start time, run distance in miles, amount of time the run took in HH:MM:SS format, description</td>
<td>Critical</td>
</tr>
<tr>
<td>4</td>
<td>Application will calculate statistics such as speed</td>
<td>Critical</td>
</tr>
<tr>
<td>5</td>
<td>Application will calculate caloric statistics</td>
<td>Critical</td>
</tr>
<tr>
<td>6</td>
<td>User can store heart rate statistics such as starting HR, ending HR, max HR</td>
<td>Like to Have</td>
</tr>
<tr>
<td>7</td>
<td>User can create profile including age, weight, height, sex</td>
<td>Important</td>
</tr>
<tr>
<td>8</td>
<td>Application statistics include profile data</td>
<td>Important</td>
</tr>
<tr>
<td>9</td>
<td>User can switch between miles, kilometers</td>
<td>Like to Have</td>
</tr>
<tr>
<td>10</td>
<td>User can prepare a race plan for upcoming events</td>
<td>Like to Have</td>
</tr>
<tr>
<td>11</td>
<td>User can store weather information</td>
<td>Like to Have</td>
</tr>
<tr>
<td>12</td>
<td>User can store route info in “Name of route” and optional “Description”</td>
<td>Important</td>
</tr>
<tr>
<td>13</td>
<td>User can search for all runs on a given route</td>
<td>Like to Have</td>
</tr>
<tr>
<td>14</td>
<td>User can enter GPS/lat./long. coordinates for starting and ending route</td>
<td>Like to Have</td>
</tr>
<tr>
<td>15</td>
<td>Application allows daily, weekly, monthly reporting</td>
<td>Critical</td>
</tr>
<tr>
<td>16</td>
<td>Application allows custom date range reporting</td>
<td>Like to Have</td>
</tr>
<tr>
<td>17</td>
<td>Application features a calendar-style interface for reports on a given day/range</td>
<td>Like to Have</td>
</tr>
<tr>
<td>18</td>
<td>User can track injury information (when occurred, when healed, what it was)</td>
<td>Important</td>
</tr>
<tr>
<td>19</td>
<td>User can view run information in calendar-style with icons/colors for injuries, best times, worst times</td>
<td>Like to Have</td>
</tr>
<tr>
<td>20</td>
<td>Application has a drop down of previously entered routes when user is entering new run info</td>
<td>Like to Have</td>
</tr>
<tr>
<td>21</td>
<td>User can view progress report in grid-style format</td>
<td>Critical</td>
</tr>
<tr>
<td>22</td>
<td>User can view progress report in chart-style format</td>
<td>Like to Have</td>
</tr>
<tr>
<td>23</td>
<td>User can backup and restore database through the program</td>
<td>Critical</td>
</tr>
<tr>
<td>24</td>
<td>User can export progress reports to Excel</td>
<td>Important</td>
</tr>
<tr>
<td>25</td>
<td>User can email reports to other users of the program</td>
<td>Like to Have</td>
</tr>
<tr>
<td>26</td>
<td>User can post recent run information to Twitter</td>
<td>Important</td>
</tr>
<tr>
<td>27</td>
<td>User can post recent run information to Facebook</td>
<td>Important</td>
</tr>
</tbody>
</table>
About this sample feature list

Notice that some features could have gone on the same line yet they are separated:

- User can store data for run date, run start time, run distance in miles, amount of time the run took in HH:MM:SS format, description
- User can store route information in “Name of route” and an optional “Description”

These are separate in our feature list because the application would not be a run tracking application without the first but could be a run tracking application if it stored the first yet did not store the second. In other words, the first is a “Critical” feature while the second is either an “Important” or a “Like to Have” feature.

Once you start writing out your feature list, you will also see dependencies at work: obviously you cannot develop the “User can search for all runs on a given route” unless you previously have developed the “User can store route info in ‘Name of route’ and optional ‘Description’” part.

You will also notice specifics:

- User can store data for run date, run start time, run distance in miles, amount of time the run took in HH:MM:SS format, description
- User can switch between miles, kilometers

This allows us to start at the critical level and expand the app later if we want to or have additional time. In the above, we were specific about the amount of time it took to complete the run (including both the format we allow and that we allow the user to enter the seconds so they can track shorter runs like 1-mile or 500k).

Other entries are very high-level and not fleshed out:

- User can prepare a race plan for upcoming events

This might mean any of 100 different things depending on what the event is and/or what a “race plan” means to the programmer. You’ll have to figure that out if and when you implement that particular feature.
Appendix D: LearnItFirst Coupon

Save 15% off the purchase of any of LearnItFirst.com’s programming courses with this promotion code. Valid for first-time purchases only and one use per customer, please.

LearnItFirst Coupon

To help you get started on the path to becoming a successful programmer, take advantage of this 15% off coupon for any LearnItFirst.com video training course.

- Coupon code: CSHARPBOOKM4
- Expires: December 31, 2009 11:59PM CST

Who is LearnItFirst.com?

LearnItFirst's online video training courses are a cross between a "live" instructor-led class and a "how to" book. You get the affordability and portability of a book and the "Watch and Learn" of having a live instructor. Because LearnItFirst features more than 2,000 online training videos, you can be certain that you are getting the complete experience.

Great Video Quality

All of our courses feature full-screen resolution videos with high quality audio to ensure that your training experience is the very best possible.

Extremely Cost Effective

LearnItFirst training courses are on par with "live" training yet you only pay a fraction of the cost. Enterprise customers can also add custom exams, certifications, mentorship programs and more.

For More Information

To learn more about LearnItFirst.com, visit http://www.learnitfirst.com/ or call +1(877) 630-6708