Recruiting Fundamentals Training

Development Languages
An Overview

IT 4 Recruiters

Overview

Development languages are as old as computers. These languages are the mechanisms that we use to tell computers what we want them to do. This document is going to provide a high level view of the types of development languages in use today, some of their history, and the ways they relate. Once we have this foundation setup we will move into discussions about how to find and hire talent for developer positions. The history will be focused on the way languages have evolved and spawned successors so that later discussions on development language families will be easier to understand.

The details of many of the languages covered here will be addressed in future IT 4 Recruiters topics, but once you have completed this course you should have an idea of the differences and similarities (at a high level) among major languages in use today. You should also be able to apply that knowledge to the hiring process for development resources to make the process move smoother, faster, and result in the best candidate accepting an offer.

Keywords

Here are some useful definitions to keep in mind while reading this document:

- Object Oriented Programming (OOP): A development approach that codes for "objects." This approach creates objects using code and the objects have properties and methods that are used to develop applications.

- Functional: A development approach that codes for functions. This approach creates tasks the computer performs based on input and provides defined output (functions).

- Batch: Development languages used for batch processing. The batch approach is very much like a list of steps that need to be performed. Each step is defined using a batch language as well as the order in which the steps are to be performed.

- Script: Sometimes considered simpler languages, script languages are ways to communicate with the operating system or environment and perform related tasks. Script languages are often used in batch processes and may be seen as batch languages although scripting can include other approaches to processing.

- Compiled/Compiler: Languages that convert the code into a set of instructions that are more easily read by the computer. Compilers convert code into a binary form that is specific to the platform that will execute the code. This often makes for faster processing and smaller storage space for the end application.

- Interpreter/Interpreted: Considered an opposite of compiled languages, these process the code as it is being executed. This takes more time than compiled code during execution, but does allow code to be generated on the fly and provides more flexibility in how the application adjusts to the tasks as they are being done.

- Cross Platform: Code that can be written for multiple platforms. It requires a compile step on each platform, but in theory the code itself does not need to change. This allows (for example) a C program to be compiled and run on an Apple Mac and then compiled and run on a Windows XP machine without changing the code.

Development Languages Snapshot

There are literally dozens, probably hundreds of development languages in use today. Some are used by millions of developers for all sorts of tasks, while some are niche languages used by a few for very specific tasks. There are languages that we hear about every day such as: Java, C# (c-sharp, not c-hashtag), PHP, and C++, but these are far from the only ones used daily. There are also numerous languages that are used to customize and/or build applications we use every day where the product name is familiar, but probably not the language such as Apex (salesforce), VBA (MS office applications), SQL (databases), and Swift (Apple iOS applications).

There are literally hundreds of development languages that you may come across in the IT world.

Developers might spend their entire career working with one or two languages, but most will become proficient in closer to a dozen through the course of their career. It is not uncommon to find senior developers with experience in twenty or more languages. This may seem like a mind boggling amount of information for a developer to have in their head. In reality, it turns out the similarities in languages make it often easy for developers to translate experience in one language to another that is similar. When development languages are seen more as families of, rather than individual, languages they become much easier to understand and relate to. All languages have some core concepts they share and then the families of languages add additional concepts that are shared by the languages in that family. The more languages one learns, the more the differences resemble dialects rather than whole new languages.

Developers often specialize on one or a few languages, but will often have some experience with several languages over the course of their career.

Core Development Language Concepts

Assignment	Set a value to another value. e.g. A = 12
Comparison	Compare values. e.g. A < 10
Logic	Combine comparisons e.g. A < 10 and B < 5 or C = 2
Grouping commands	Define a unit of work. e.g. “Make a sandwich” is done by 1. getting bread, 2. getting meat, 3. getting condiments, and 4. Put the meat and condiments between the bread slices.
Relative tasks	Perform a task based on an input. e.g. if the dishes are dirty then run the dish washer

Once a developer grasps core concepts it becomes easier for them to apply those concepts in new languages.

The families of development languages are similar to how spoken and written languages have families. Anyone who has spent time with Spanish, for example, will find Italian very familiar. This familiarity expands to a number of languages when you consider the romantic languages: French and Portuguese, among many others. The latin roots of words in these languages make it easier to translate a concept into words in a specific language. For example, the concept of "feeling good" shows across a number of languages in words with buon, bene, boon, bon and similar combinations included as part of the word (benefactor, buenos dios, buona notte, etc). In the world of development languages this can be seen in the example of the concept of "show something on the screen". Almost all languages support this action and it often is coded with a command to "print" or "display". The most common commands in development are very similar across languages including: if statements (if), value assignment (=), loop commands (for,while), and "perform a task based on a value" (case,switch).

Development languages have similarities among them just as other spoken and written languages do. This makes learning successive languages easier.

If statements, looping constructs, value assignment and logical constructs are some of the most core concepts among development languages.

Development languages are like most things in IT. Once you understand the underlying concepts, the implementation is a matter of semantics. Put simply, once you know what questions to ask, it is just a matter of knowing how to ask it. We see this in spoken languages as well. That is why we often learn common questions like: what is your name?, where is the bathroom?, and can I please have a beer/wine? very early in foreign language education. I don't need to know anything about Chinese on a whole in order to ask some one how I say "What is your name" in Mandarin. That is why phrase books are so effective when you need to communicate in a foreign language. Development languages use these same core concepts and can allow a developer to "get around" in code in a language they do not know.

History

Computer languages started as turning things on or off. It was modeled this as a one (on) or a zero (off). This is known as binary coding and it still exists today although there is so much abstraction provided for development that almost no one really writes code at that level. There are tools for writing code to that detail. A slight step up is machine language which can move things around in memory and set and clear values of several switches at once. Machine language is very specific to the hardware it is being written for so it is not portable to other types of hardware. For example, machine language code for an apple

The zero and one representation you often see showing computer programming or processing is a reference to computers being a series of switches turned on or off.

iPhone 5 will run on other iPhone 5 devices, but will be different for other devices including an iPhone 5s or iPhone 4. Luckily, we no longer need to worry down to this level of detail. There are compilers and code generators that handle the tedious work. Although not completely irrelevant, for our purposes machine language is exactly that: a language only used by machines. Lets move on to some languages one might run into today.

There is a lot of detail that modern languages hide and we do not need to know those details.

In the late 1950s the start of modern languages began with the creation of FORTRAN, COBOL, and LISP. These languages are referred to as third generation languages to distinguish them from machine and binary languages. These are the oldest examples of languages still in regular use today. These languages provided an easier way to write code by providing a layer above machine language that was more human readable. A sample is shown below of FORTRAN code, it has been made more readable over the years, but, as you can see from the example, it is a long way from easy to read. LISP and COBOL are typically less readable by non developers.

It can be argued that FORTRAN, COBOL, and/or LISP is the ancestor of any language today although none of these is commonly used today.

Sample Fortran Code:

```
initialize random numbers
    seed = 35791246
    call srand (seed)
    do j= 1,100
      niter = niter+100
      count =0
      do i=1,niter
        x=rand()
        y=rand()
        z= x*x +y*y
        if (z .le. 1) count =count+1
      end do
      pi(j)= count/niter*4.
      write(*,10) niter,pi(j)
      format('Number of trials is: 'i5,'  estimate of pi is:',f8.5)
    end do
    end
```

What we do see in these first three languages is the start of types of languages. Put simply, LISP works on lists, FORTRAN is functional, and COBOL is data oriented. This ability to categorize languages becomes important as we look at other languages and draw similarities between them.

We see language specialization start even in the comparison of COBOL, FORTRAN, and LISP.

In the early nineteen seventies we see the Pascal and C languages created, with BASIC following not long after. These are also third generation languages (3GLs) and are aimed at being cross platform so that it was easier for developers to apply coding skills and experience across a number of machines. The code written in these languages is somewhat more human readable than the earlier 3GL languages. Pascal and C are still in heavy use today although barely recognizable from the forms they started in.

Pascal and C are later 3GL languages that are still in use today in some form or another.

BASIC has been supplanted by Visual Basic and then Visual Basic.NET as it evolved into something similar to the popular languages of the time. Pascal rapidly evolved into Object Pascal and then into Delphi as object oriented and the fourth generation languages became popular. There are a number of ways fourth generation languages have been defined, but the common view is that a 4GL gets away from manipulating bits and bytes into more abstract commands, making the code much more human readable.

Fourth generation languages abstract coding another level and often provide point and click methods of code creation or development.

Towards the end of the seventies the U.S. department of defense adopted ADA as a primary programming language for their projects. ADA is another 3GL that took strengths from a number of other languages, including C and Pascal, to make it easier to learn while still being a valuable and versatile language. It is still in use today, even though other languages have replaced it in modern projects. When you see ADA, think Department of Defense and other government entities.

When you see ADA think Department of Defense. It is still in use and often with applications that were started decades ago.

The eighties saw the rise of object oriented programming. C++ and objective C were created to embrace this new way of developing. Both of these languages were built on top of C and provided a way to transition C developers to the object oriented successors. C++ ended up the far more popular language of the two, but both are still in wide use today.

C++ and objective C were created to add object oriented support to the C language.

A challenger in the object oriented world was Smalltalk. It appeared around 1980 and was the first “pure” object oriented language and spawned a number of similar languages including eiffel, but all of these have faded into a form of obscurity. Modern projects occasionally pop up with a need for these sorts of skills, but they tend to be more prevalent in European development shops where adoption was greater. The popularity of Java ended up causing the demise of Smalltalk as it provided an object oriented approach with less overhead than Smalltalk and similar languages. It was also easier to learn for functional developers. Java had (and has) similarities to C++ which makes it easier to transition from C/C++ to Java than to Smalltalk.

Smalltalk is a purely object oriented language that fell into obscurity as Java rose in popularity.

As the Eighties progressed, processors and memory became more powerful while costing less. This made interpreted languages more popular. Compiled languages such as C and C++ would translate code into machine language before the application was run. This makes the application run faster (it did not have to spend time translating the code), but it is not quite as flexible in functionality. Compiled code that functions based on aspects of the data is hard to write. Even in modern times, compiled code is often used in the most time sensitive programs such as real time communication and video processing, or environments that are very space or processor conscious such as small/micro devices.

The Eighties saw heavy use of compiled code and applications due to universal constraints on application memory and storage space. Networked applications were very rare.

Interpreted languages perform the translation to machine language while the application is running, this allows code to be created as the application is running. This dynamic code aspect allowed applications to be more flexible and solve new problems.

The early Nineties saw the introduction of the Internet and better network connection speeds. Development languages embraced this and tended to have better support for communication, as well as being more likely to be interpreted. PERL was created in the late Eighties as a general programming language that was easier to use than many shell and batch languages, but it is probably best known as the predecessor to PHP (created around 1995). The early Nineties also saw Python, C#, Java (and soon after: javascript), and Ruby created, all of which are sometimes referred to as web languages due to their impact on the applications of the world wide web and being more commonly used for web applications than traditional desktop or server applications.

We saw a rise in interpreted languages and applications in the Nineties as memory and storage space prices dropped while processing power increased.

These languages tend to be much more human readable and saw a rapid adoption due to the ease of access to training materials as the Internet grew in popularity. Professional training and certification options became popular around this time in a way that was more accessible to the typical developer which also sped adoption. The tech bubble of the late Nineties saw a large amount of money going into training and other educational efforts that helped create a large population of developers educated about a large number of languages and able to learn new languages seemingly on a whim. There are arguments about a corresponding lack of quality in developed applications, but that is for another training course. When we talk about language families the web languages all share some traits even though they also typically share traits with LISP, COBOL, or FORTRAN.

Modern languages that started in the late Eighties and beyond typically are much more human readable than earlier languages in an attempt to ease cost of entry into development.

```
public class SimpleWordCounter {

    public static void main(String[] args) {
        try {
            File f = new File("ciaFactBook2008.txt");
            Scanner sc;
            sc = new Scanner(f);
            // sc.useDelimiter("[^a-zA-Z']+");
            Map<String, Integer> wordCount = new TreeMap<String, Integer>();
            while(sc.hasNext()) {
                String word = sc.next();
                if(!wordCount.containsKey(word))
                    wordCount.put(word, 1);
                else
                    wordCount.put(word, wordCount.get(word) + 1);
            }
            // show results
            for(String word : wordCount.keySet())
                System.out.println(word + " " + wordCount.get(word));
            System.out.println(wordCount.size());
        }
        catch(IOException e) {
            System.out.println("Unable to read from file.");
        }
    }
}
```

Shown Above: Sample Java Code

Although Java is still a few steps away from being easy to read for non developers, the usage and coding standards attempt to make it more human readable even while adding new, complex features.

As language bloodlines go, PERL/PHP is more of a functional language like C and Pascal although it does support OOP. Java has strong C roots and a C++ influence. Ruby and Python have an almost LISP like feel although they are strongly object oriented in nature. C# is possibly closest to a C like language with strong influence from Java. It is a general purpose language that can be used to build compiled stand alone applications, client server applications and even the most complex web service oriented distributed applications. The only thing that stops Java and C# from being direct competitors is that java is cross platform while C# is limited in where it can be run. We will look a little closer at these bloodlines in the section on cross-over skills.

Microsoft is using for .NET to be more cross platform friendly and has made it more and more so with each release.

The Nineties included an attempt at “write once, run anywhere” that started with Java and is seen in other languages of the time up to current day. This is one of the strengths of interpreted languages: code can be written, and then the source can be copied to another device and runs the same. This is not something that has been 100% achieved, but it is “close enough.” There are a number of popular programs today that are not compiled for every device/system, but are instead written, packaged, and then able to run on nearly any system. Java is the most popular language for these applications, but Ruby, PHP and

Write once, Run anywhere was a Java catch phrase/slogan early on and many languages have adopted that goal to varying degrees of success.

Python (among others) are also being used for this purpose on millions of machines.

The twenty-first century has not brought many new languages as much as it has libraries and extensions of existing languages. Javascript, in particular, is the code that launched a thousand spin offs when you look at all of the popular skills today. JQuery, Dojo, Angular, and many other skills out there are javascript at their foundation. Java has also seen a number of libraries get treated as full blown languages in their own right. Hibernate, Spring, Struts, and numerous other popular libraries have a java core. There is not a term fifth generation language, but we are essentially seeing those today as there are languages built on top of languages and they often provide quick ways to perform tasks in the core language. It is almost becoming mandatory that languages include a “new improved” version or add-on released as we have seen with Ruby (Rails), Python (Django), and PHP (nuke, Zend, cake, and others). These libraries and frameworks provide even easier entry into development jobs provided the development does not need to delve into the underlying language.

In the last ten or so years languages have grown through extensions and new libraries rather than new languages.

Common Extensions and their core tech:

- Rails-Ruby
- Django-Python
- Zend-PHP
- Nuke-PHP
- Cake-PHP
- Spring-Java

As you can see in the code samples provided, the latest languages and libraries are not as concerned about human readable code. There is a trend towards code that is easier for a machine to interpret and smaller code size. Applications are often downloaded these days and web applications send code across the network to be run on a browser so the smaller code size reduces those load times. There is even a process called minimizing that is becoming popular and it reduces code size, while making it easier for a machine to read, but at the cost of human readability. It has been found to be a sort of middle ground between compiled code and interpreted code.

Code minimizing is a process that reduces white space and shrinks names where possible to make a small file size, but at the cost of human readability.

```
<script>
$(document).ready(function(){
   $("#btn1").click(function(){
      alert("Text: " + $("#test").text());
   });
   $("#btn2").click(function(){
      alert("HTML: " + $("#test").html());
   });
});
</script>
```

Shown Above: Sample JQuery Code

Modern Landscape

When you look at popularity of languages today (2016) Java is the top language with C and C++ right behind it (IEEE: http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages). Python and C# round out the top 5. The members and order of the top five languages has been pretty stable for the last several years. These languages generally can be used anywhere, and for any type of application, which is a factor in their popularity. The next five in popularity: R, PHP, Javascript, Ruby, and Matlab are somewhat stable over the last few years with R being a newcomer and the precise ranking of the five varying by year. R, Javascript, and Matlab are "niche" languages and their rankings are impacted by the types of applications they are best suited for.

The top five languages over the last few years:

- Java
- C
- C++
- Python
- C# (C-Sharp)

As one would expect, the majority of jobs posted will include Java, C/C#/C++, or Python. The confusing part is that a lot of modern positions include a desire for skills in other languages as well. The single language environment is becoming more and more rare as secondary and scripting languages increase in popularity and are easier to use. The increased use of contractors and frequency of acquisitions has also contributed to this IT diversity.

Positions often require experience in multiple languages as IT shops grow in diversity through acquisitions, side projects, and one off projects that end up in production.

There are libraries and frameworks used by a large number of developers that require knowledge of one of the top 10 languages, but the core language may not even appear on the job description. For example, job positions may be posted that include a desire for Spring experience, but Java is not mentioned, or .NET and neither C# nor Visual BASIC are mentioned. This can make it harder to find candidates with complementary skills particularly when new or cutting edge libraries/frameworks are being used.

Be aware of framework and library experience that does not mention the core language.

Common Technology Pairings (language : framework)	
C, C++ : STL	Java : Struts
C# : NStruts	Java : Swing
C# : WCF	Java, C# : Hibernate
C#, Visual Basic : .NET	PHP : Code Igniter
Java : Google Web Toolkit	PHP : Nuke
Java : Grails	PHP : Zend
Java : JSF	Python : Django
Java : Spring	Ruby : Rails

Test candidates on core languages skills as well as framework knowledge to find better candidates.

This rise in frameworks and code generators has not only caused job listings to often focus only on the framework or code generator needed, it has also made it more common to find developers that know the tools, but not the language. We will look into this in more detail in the sections on development language conversations and screening questions.

We look at language classifications as a way to search for candidates based on general experience rather than a specific skill.

Web developer jobs are arguably the most common types of jobs posted. Although not always mentioned as such, Javascript and HTML are default web languages and are used on almost every web development project. A web developer that does not have these skills to some extent (at least exposure) is NOT a web developer.

Language Classifications

There are a number of ways to classify development languages, but most of these classifications are more focused on the languages from a technical point of view, rather than similarity for learning purposes. When screening potential candidates for a position, it can be more useful to know what sort of languages will make it easy to learn the position requirements, rather than a technical overview. This changes the candidate pool to candidates that are better suited for the position in the long run through practical experience, rather than one that can match a list of credentials. With that in mind lets look at a classification of languages that is focused on ease of transition to new ones so we can see where complementary skills might help you find a better candidate than a direct match.

Use language classifications as a way to increase your candidate pool, particularly with hard to find or niche languages.

An overall note about transitioning to a new language: Every time a new language is learned it will be easier to make the transition. A developer with twenty years of experience in a single language will probably find it harder to transition to any language than a developer with five years of experience in three (or more) languages. This progressive ease of learning a new language is due to a few reasons. First, as one learns more languages they will be exposed to more approaches to solving problems. This is the same concept of someone that only knows how to use a hammer sees everything as a nail. Second, each additional language adds another set of commands and formats for doing typical tasks. This leads to successive languages being more likely to be familiar in syntax, style, or both. For an example we can look at spoken and written languages again. As one learns more languages there is more likelihood that a word in a new language is similar to, or the same as, that word in a language

The more languages one knows, the easier it is to learn another one and the faster one can become productive in that new language.

that is already known (mama, papa, etc.). Key concepts may appear again as new languages are learned such as the gender of words seen in Spanish, French, Italian, etc. or words having different tenses.

C is one of the most complex and foundational languages in use. It is sort of unique in the number of concepts developers are typically exposed to. It is hard to switch to C from other languages, but it is one that is easily transitioned away from. A sports team analogy would be to think of most languages as the skills to play a position or two, while C is closer to the ability to have all of the skills used in the sport. A C developer can typically become at least somewhat productive very quickly when moving to another language. There are very few concepts in other languages that are not also addressed in C, particularly when you lump C++ in with it. This is part of why so many computer science programs include C as one of the first and main languages students will learn while pursuing their degree. Object Oriented concepts tend to be the biggest step a C developer needs to make when moving to another language.

C is one of the best languages to have on a resume because once one has learned C they have at least been exposed to a large amount of programming concepts.

There are a number of languages that can be boiled down to "C with object oriented added and some of the 'harder' parts of C removed." All of these languages are fairly easy to learn from C and also transitioning among them is not too difficult. These languages include C#, Objective C, C++, and Java. Java and C# are more networking and Internet "friendly" so they are easy to transition between. Objective C and C++ are slightly more complex and thus a little harder to transition between.

The next great concept to have experience with on top of C is object oriented. This combination allows one to relate to nearly every modern language.

In many of these core languages a number of libraries have been created that are almost identical across these languages making it even easier to transition. Most of these tools are open source and we will spend more time reviewing those in the course on open source.

Sometimes libraries and frameworks make a move to a new language easier and more likely to be successful.

Scripting languages are not as similar amongst themselves as the C based languages, but there are a lot of similar concepts that make transitioning among them fairly easy. These include awk, sed, Perl, Ruby, windows batch, shell scripts, Ant, Make, Rexx, Tcl, and many others. It is easy to move to these from almost any other language, but the move from script languages to others is a big step and not always successful.

Moving to a scripting language from the "big 5" is fairly easy, but moving from scripting languages is not always successful due to vast differences in complexity.

Script languages are often simple enough in complexity and syntax that anyone can be taught enough to do some amount of scripting, but a developer that only knows script languages is arguably not a developer. They are better classified as an entry level programmer or coder.

Adoption

We have already looked at popularity of the languages and also mentioned that modern jobs often involve multiple languages or frameworks. Some of the pairings that exist are because one member of the pair requires the other and some are due to the type of applications to be built.

Javascript and HTML are used throughout the web and show up on almost every web development position that is beyond junior level. PHP, Java, .NET (C# and Visual Basic), Python, Ruby, and other web friendly languages may be used for some of a web application, but HTML and Javascript often are listed as required or highly desirable. HTML code can be written by itself, but I can not think of a situation where Javascript is used and HTML is not.

Web developers will always at least have exposure to HTML, CSS, and javascript even after only a year or two of experience.

CSS (Cascading Style Sheets) often will be paired up with HTML and Javascript on any web development project, but CSS is not really a language in itself. It is a script for building design templates for web pages and has mechanics closer to doing layouts for print media than development languages. Knowledge of CSS is almost always helpful for web projects, but it often falls into the graphic or web designer toolbox more than a developer. If a developer position requires CSS knowledge (particularly if it requires CSS design experience) it is going to be front-end (user facing) development. These roles may be better suited for a user experience specialist even if only as a short term contract to handle the front end design work.

CSS expertise often is a mark of a UI designer over a developer and similarly jobs that list CSS as a primary requirement are much more likely to be for a designer than coder.

As pairings go, there are a number of frameworks built on languages. In these cases, the framework always implies knowledge of, and experience with, the core language being a nice to have at least and is most often required. In some lower skilled roles (mostly entry level), the framework experience alone will be enough, but at mid level or higher experience levels there will always be an expectation of experience in the core language as well. Refer back to the table above for a partial list of pairings focused on the most common ones seen today.

A framework listing without its core knowledge is always an indicator that clarification of the role or position is needed.

From the table, there are some areas where clarification will help. For the Java pairings, core Java is always going to be assumed at a somewhat equivalent level and maybe even slightly greater. For example, a requirement of five years of Spring experience typically implies at least five to eight years of java experience. Languages outside of C, C++, Java, and C# are often viewed only by their framework since the core language is so rarely used without including the framework. Examples of this include Ruby-Rails and Python-Django.

Language - framework pairings often assume a slightly higher experience exists in the core language than the framework.

It is important to note also that Microsoft's development framework, .NET, is built so that multiple languages can be used with it. C# and Visual BASIC are the most common languages, but there are a few others in use today and in the past. .NET implies one of the implementation languages will be required, and usually a customer will expect a specific one. For example, a company may list a need for 10 years of .NET experience, but further discussion will turn up that they expect the 10 years in C# or Visual BASIC. This is not always the case, as .NET components can be written in either language and the customer may have needs that are language agnostic. They might even desire candidates to have experience in multiple .NET languages. In a similar fashion C# and Visual BASIC often are assumed to be used to create .NET code, but there are environments where the languages are used and .NET is not needed. Once again, when there might be doubt: ask or clarify.

C# and Visual BASIC are potential implementation languages when .NET is mentioned as a requirement.

There are a number of open source libraries that started in the C or Java world that have been ported to C# or PHP (or vice versa) and the names are similar but not identical, presumably to help avoid confusion. Most C# libraries start with an "n" so NStruts, NHibernate, Nunit, etc. PHP ports are not always as simple but they typically start with "p" or "php" e.g. Struts for PHP, phpunit, etc. Java libraries often contain a "j" e.g. jUnit, log4J, etc.

College graduates with a computer science degree are an area ripe with "diamonds in the rough."

Another point of confusion can be that there are often code names for these development languages. Always do a quick Google search when you come across something new. One set of code names that are disappearing (but still may show up on position descriptions) are J2EE, J2SE, and J2ME used for naming specific Java language sets. J2EE was for web applications, J2SE was for Java applications in general, and J2ME was for mobile applications.

A lot of big and successful companies like Google, Microsoft, and Amazon will find hires that they can train up to senior level developers over time.

Market skill set

Developers and Programmers often start out focused on a single language until they reach a mid level, or greater, amount of experience. They will typically be exposed to, and possibly have solid experience in, other languages, but they will have a primary language in which they are most comfortable and capable. As you move into the senior developer ranks and beyond there may still be a strength, but often there are close second and third languages that are used to develop similar applications. For example, a senior developer in Java might easily have several years of C# and PHP experience due to the large number of web applications they have built. This is more prevalent in developers with a consulting background. Even large companies have code language shifts over the last decade or two, and long term employees went through those shifts as well.

Entry level coders typically have experience in only one or two languages, but veteran developers may have dozens.

Junior developers with a college degree are usually going to have been exposed to at least three or four languages. They will have the expected junior level developer knowledge in each of those languages, but will still tend to have a single language that is their strongest. This often helps developers with college degrees advance faster. They already have been exposed to a number of concepts that make transitioning to other languages (and learning the latest hot language) much easier when compared to those with a more narrower knowledge of development.

College graduates with a computer science degree often advance faster due to solid knowledge of CS theory.

There are a large numbers of developers available for the most common languages, but as you go down the popularity list and into obscure languages it is harder to find resources. When looking for developers, it is always important to consider geography. There are major cities, regions, and countries that have developer numbers skewed towards certain languages. The salary expectations also shift based on geography so sometimes it is possible to find a niche developer at a great price. This is more likely when remote work is allowed, or if a developer is looking for a move out of an area flooded with talent (allowing them to become a big fish in a little pond). It may also be near impossible to bring in talent when there are too many higher paying areas that have a need for developers experienced in the desired skill set. Developers tend to flow to the areas where their skills are needed the most.

Popular languages have more developer available and niche language requirements can be very hard to fill.

Supply and Demand of resources vary by many factors including industry and geography.

Entry level positions almost always can be filled with a strong candidate simply by looking for recent graduates. Once you get into higher experience levels it may help to search for candidates in certain industries. There is a shift in popularity of languages as

Development Position Types	
Front End	Develops application user interfaces. This includes creating and coding the parts of an application that the user interacts with. High end developer positions focused on front end development may even look for User Experience skills.
Back End	Also referred to as Database developers in some cases, these developers tend to write code that interacts with databases and other data storage types.
Middle Tier	The middle tier is also referred to as Business Rules or Business Logic portion of an application and the core functionality of the application often exists here. This is where data is manipulated and calculations are made to provide results to the user.
Data Analysis	This is coding that is purely data manipulation and calculations.
Scientific	As it implies, this is coding involves scientific or engineering calculations. It is a different skill set than Data Analysis.
AI/Learning Systems	Artificial intelligence/Expert Systems/Learning Systems coding that alters how a program runs based on data the application has "learned"
Mobile Applications	Small device coding including phones, tablets, watches, etc
Shrink-Wrap	Commercial software development where the application is sold directly to a customer as part of the business model.
Internal	Coding where the end product is used within the company. The primary customers for the developed product are all internal staff of the same employer as the coders.

A position should always provide enough information to make it clear the type of position that is to be filled.

Developers prefer and specialize in types so a match to type can be a deciding factor when an offer is made.

Some developer types such as AI and Scientific are niche applications and may be harder to resource.

you look at different lines of business, so looking in other industries may help find a skilled developer that can be trained in the new (to them) business. You just need to be aware that there might be a ramp up cost for entering the new industry.

Startups are a great source of developers that are more likely to have wider development exposure. The lack of resources available to a typical start-up often push the developers to learn more skills outside of their primary language including wider business knowledge. It never hurts to try to raid a failed start-up for good talent.

Startups are often a source of better than average talent when comparing years of experience.

Certifications

There are numerous certifications available for almost every language. The more popular languages will have vendor and third party certifications available at several levels to help assess skills from beginner up to veteran. These certifications are great ways to vouch for for levels of knowledge and usages for a language and/or framework. For example, Java certifications include programmer, master, architect, web and other variations.

The number of certifications can be mind boggling, and are hard to keep up with, so they are not often seen as a specific job requirement. It is more common to see certifications as a plus or a "nice to have." This being the case, presenting a candidate with any certification often helps open doors and it is worth looking into the certifications that will most likely apply to positions you are trying to fill. A hiring manager that knows about relevant certifications can vet talent faster and use the skills the certifications test as a source for screening questions. In some cases it is as simple as starting with a list of skills required for a certification and then prefixing the skill requirement with "Tell me what you know about".

Certifications are often nice to have items on a development job posting rather than a firm requirement.

Some certification programs are simple and not much more involved than paying some money and reading a few articles. These are not much different from the old "mail order degree" scams. These are rare. The majority of the "top 5" language certifications include a reasonable amount of training and/or education. There is also a non-trivial exam (sometimes multiple exams) that must be passed to receive the desired certification.

Look at the vendor that supplies a certification to ensure the worth of the certification.

Certifications can be worth more than a degree in some cases, particularly junior to mid level positions, and should be highlighted when possible. A manager can help their retention numbers by finding some training budget dollars to use on certifying developers once they have been hired. It is a great incentive plan, and it helps ensure your team is up to date with the latest technology changes. Certifications have more practical use than information gathered in conferences and the knowledge is more likely to be retained due to the testing aspect of a certification. Some certifications even include implementation requirements (a project must be built and submitted) and those can greatly boost your confidence that the certified developer can hit the ground running.

The value of certifications is such that you can sometimes replace traditional degree requirements with a few relevant certifications.

Suggested Developer Certification Sites	
Java	http://education.oracle.com/pls/web_prod-plq-dad/ou_product_category.getPillarPage?p_pillar_id=5
C#	http://www.newhorizons.com/courses/microsoft/visual-studio-training.aspx
PHP	http://www.w3schools.com/cert/cert_php.asp, http://www.zend.com/en/services/certification Note: As an open source technology PHP has several certification options.
C/C++	http://www.tomsitpro.com/articles/programming-certifications,2-274-6.html
Others	http://blog.pluralsight.com/top-7-programming-language-certifications

After all this happy talk about certifications it is important to note that certifications show a base level of knowledge and should not be blindly used to replace experience. Certification paths can be much better at teaching a technology than a more general computer science degree. In the long run, the developer that knows programming approaches and theory will usually be able to outperform one that simply knows a language to a deeper degree. Consider the short and long term goals for a position when weighing the value of certifications possessed by a candidate.

Experience should always trump degrees and certifications.

Development Language Conversations

Development languages can be tough to get your arms around even when we try to keep the scope to an overview. The good news is that you do not have to get your arms around all the languages out there (or even the top ten) in order to find the best developers for the job. We now have enough foundational knowledge to move forward and look at how to best define developer positions and to screen candidates.

Clarifying Questions

The obvious questions about development positions start with the languages to develop in, but that is just scratching the surface. The type of development is always the best place to start when clarifying a development position. There are a number of development types and characteristics of each type. The table below provides a quick overview of the types and the traits. These are not hard definitions, nor are they comprehensive, but it should help provide some clarity into most development positions. Once you have a language and a type it provides for great questions including: “How do you see that language as a good or bad fit for that type of development?” When we look at cross over and complementary skills the development type is sometimes as important as the language itself for placing a candidate in the best job (for both them and the company).

Always clarify the type of development that is needed and the role the position will play.

These development types may not mean much to the recruiter or manager, but they do to the potential employee. Entry level developers may not know what these development types are, but as developers gain experience they will be aware of the types of development positions in the market and tend to gravitate towards one or more. The skill sets a developer has will also make them a stronger or weaker match for a position based on its type.

If you struggle to determine the development type, ask a developer for help.

Beyond the position type, the next important facet of a development position is the environment and expectations. This is best expressed as a number of questions that the hiring manager or recruiter should be able to answer for the candidate:

- How big is the team?

- What roles are represented on the team? (QA?, Database specialist?, configuration management?, tech writer?, etc)

Development positions are often highly impacted by team and environment so include those in any job description.

- How is the team structured? (Flat?, Single manager?, Manager plus team leads?, etc)
- What sort of customer involvement is expected?
- Will customer support be required? On call required?
- What are the typical tools used/available?

Typical Problems Faced Summary	
Front End	Number of steps (clicks) to perform a task. Communicating with the user (labels, messages, etc) Special data type arithmetic (dates, times, financial) Spacial programming (placing objects graphically) API Usage
Back End	Special data type arithmetic (dates, times, financial), Performance tuning, API Usage/Creation, Transactional programming, Code generation/Dynamic coding, Thread development
Middle Tier	Transactional programming Special data type arithmetic (dates, times, financial) Set related functions Complex Problem Solving
Data Analysis/ ETL	Set related functions,Formulaic programming Thread development
Scientific/ Financial	Set related functions, Formulaic programming, Spacial programming (placing objects graphically), Complex Problem Solving
AI/Learning Systems	Adaptive programming, Code generation/Dynamic, coding, Thread development, Complex Problem Solving
Mobile Applications	API Usage/Creation, Customer Support, Production Deployments, Hot fixes/Patches, Multi-Platform support/development, Spacial programming (placing objects graphically), Thread development, Full SDLC development
Shrink-Wrap	Customer Support, Develop to plan/schedule,User documentation, API Creation/Usage, Library creation, Exposure to Project Management, Production Deployments, Hot fixes/Patches, Multi-Platform support/development, Full SDLC development
Internal	Exposure to Project Management, API Creation/Usage Customer Support, Develop to plan/schedule, User documentation, Full SDLC development

Note that the typical problems faced are a great way to find complementary skills/ experience for developers. As more detail is made available for a position it will be easier to find a great fit for that position.

Some developer types are more common than others and that can be an early indicator to the difficulty in filling a position.

Market conditions will effect how important (or not) these answers are. When the market is better suited to the developer, all of these questions will have answers that can be deal breakers for the candidate. When the market is shifted towards the hiring company, candidates might not care about any of these traits of the position. This can be a problem when the market gets better (and there are other jobs to jump to), so make them part of the hiring process to improve retention.

Market conditions can cause job factors to be more or less important, but attention to these details in all conditions can contribute to improved retention rates.

The question about tools also extends to frameworks. This is an area that has become more important to developers in recent years. It is not uncommon for newer developers (less than 5 years experience) to require a substantial ramp up time when they are placed in a position that does not include using the framework they have spent the most time with. This is less a factor in candidates that have core language experience, but that core experience is becoming less common. Many entry level jobs today start with developers working with a framework to reduce their ramp-up time. This often has the side effect of it taking more years of experience for a developer to get the core language knowledge that used to come early in a development career.

Modern developers should be assessed on frameworks as well as languages to determine best fit.

In some cases even framework versions can be a critical part of the path to productivity. New concepts may be introduced in a new version and some features may be “deprecated” or removed from the framework. Candidates may ask about version numbers or names so be sure to clarify those details up front.

Version numbers of languages and frameworks can be important to matching the right candidate.

Crossover/Complementary Skills

The first area where you will tend to see common success in crossover attempts is in the area of frameworks. It is hard to find data (or even consistent anecdotal evidence) as to whether lateral cross over (new framework, same language) or vertical cross over (same framework, new language) has better results. Modern frameworks are sometimes able to provide such complete abstraction that developers may not even use the core language on a regular basis (if at all). In these highly abstract frameworks the vertical transition may not even appear to be a transition for the developer and they are fully productive on day one.

Frameworks can be a great way to help developers cross over from other language experience.

Another big source of potential complementary skills for developers is in the area of problem solving. The developer types can be used to provide a list of typical problems a developer is asked to solve (as shown in the table above) and this often provides great opportunities for complementary skill replacement. On the other hand there are developer types that have few similarities. For example, front end developers tend to spend a lot of time solving problems that back end developers do not, and vice versa. The reason that the developer type is so important to clarify is that it implies the type of transition (if any) a developer is likely to be required to make as they move to a new position.

Although we have described developer types in detail the best use of them are as generalities. Focus on similarities in problems solved.

There are complementary skills among languages, but the easiest way to view those, without going into details about the languages, is to look at the developer type for the position. Do not worry about the technical details about the problems to be solved in the table above. When you see similar problem names you can assume, for these purposes, that the problem is shared among the developer types.

The problems developers face are not always going to be part of a development position so it helps to ask (when screening) if these sort of problems are familiar to the developer. If the names and/or description are not familiar to the developer then asking for a list of typical problems the developer solved in the past will help. Likewise, clarify in the position posting (or with the hiring manager) what sort of problems are expected to be solved by the developer.

When screening a developer it helps to go into details of problems solved rather than staying up at the level of developer type.

Weasels

Development work tends to pay very well and thus attracts its share of weasels. Luckily, they tend to fall into one of two categories. The first category are weasels that have been passed by as technology changed. These weasels used to be real good at their craft, and still may know some core information, but they tend to do things because “it has always been done that way.” These weasels will struggle in environments that need to take advantage of technological advances. They may have little, or no, exposure to frameworks, but can be highly skilled in the core language. They enjoy their job, but may not be as fast as modern developers that take advantage of the latest tools and frameworks. These weasels will try to take a development group backwards instead of forwards. They will not only be knowledgable about the older, more established parts of development, they will also push to stick to those methods rather than adopt current advances. These weasels sometimes look like innovators, but they are looking backwards instead of forwards.

Weasels are either stuck in the past or trying to skip ahead to the latest hot technology with a good foundation in development.

The other weasels are the ones that have done a little development work (or a little in the given language) and are looking for a position beyond their skills and experience. They essentially look for ways to get on a fast track to a new language at the expense of experience. Sometimes, these weasels simply lack self awareness. They may have the best of intentions, or they may just be trying to use techno babble to get a better pay check (or experience in a "hotter" technology).

The good news is that shining a light on either of these weasel types just requires asking them to provide recent examples related to their development skills. The first type of weasel will talk about older technology and talk down about newer advances or tools. These weasels may remind you of the middle aged average Joe that often reminisces about the glory of their youth. They often will even claim that the old way is the best and that modern tools are too slow, too clunky, or they lead to inefficient code. They might have valid points with some of these arguments, but maybe less efficient code is an ok trade off for a dramatically reduced development effort. When faced with the facts of a new tool or framework being more productive for developers, these weasels may fall back on claiming that the facts may be true for most developers, but not for them. The "past it" weasels are often in the position they are due to fear of change, or maybe even simple laziness about keeping up with technology.

The best anti-weasel approach is to ask for details and examples of their past work.

The "stretch" weasels might have some good stories about the language or framework, but will show themselves to be bit players. They might be indirectly involved, or might even come up empty, when asked for concrete examples of more advanced work. They often will have a heavy reliance on frameworks and code generators. When tested on core code knowledge, the questions will be answered poorly or incorrectly. One way that weasels will move into a new language or framework is to go through a few tutorials or example projects in the new technology. A direct question sometimes will reveal this, but it is more reliable to ask about the development work they have done and drill down into specific applications or modules they have written. A question about how they would do that work differently based on what they learned from the project is always a great way to detect weasels. Their answers will usually be disjoint or almost dismissive (e.g. I would do nothing different).

Good stories about past experience may turn out to be cases where the weasel was only a small part of a large team. Always ask about a developer's role in a team.

Weasels might provide a great portfolio, but on deeper review the portfolio will be found to be example apps, a minimal use of a framework, or something else that looks good, but has poor or broken functionality.

Screening Questions

We discussed weasel filtering questions, and now we can move on to general questions for screening developers. These questions will help an interviewer get a feel for the type of developer, specific experience, and general development knowledge. These questions will also help provide the interviewer with enough details to determine whether a candidate is a match for the role and not just the requirements.

- What languages have you used? Which is your favorite, or the one you enjoy the most? *Note: This is a great way to get them talking about their experience and a way to find good development position fits*.

- Of the languages you have used, what is your favorite and least favorite? and why? *Note: Another good one to get them talking and going a little deeper into what their thoughts are on a language or two. This is a great question for getting a feel for how the candidate thinks.*

- How do you handle code documentation? *Note: This can show experience and attitude. Most experienced developers will provide in depth answers about either self documenting code, code generator friendly comments, or other tried and true methods. Entry and even some mid level developers will provide short answers along the lines of: they comment code when they need to (or something quick like that).*

- What is your comfort level with version control? How do you address simultaneous coding? *Note: More experienced developers will be comfortable with version control unless they have always been a lone coder. Their answer to simultaneous coding will be biased by the source control tools they have used. Senior developers will talk about multiple approaches depending on the tools used for version control and potentially even branching strategies.*

Screening Developers

- Languages used
- Favorite Language
- Developer strengths and weaknesses
- How do the handle documentation?
- What is there version control experience?
- Have they been asked to tune code?
-
- Object Oriented Experience?
- Problems they have been asked to solve
- What is their desired development type?
- What are their environment preferences?

- Have you had to tune code for size or speed? If so, what was the goal and what was your solution? *Note: Mid and Senior level developers will almost always have done this somewhere on their journey. The depth and thought that goes into this answer will also be a good indicator of the true experience level of the developer. Junior developers will list a couple of steps, while experienced developers will provide a lengthy list of tuning steps and possibly even the reasoning behind the steps. This also serves as a great question for finding out how strong or weak the candidate's communication skills are.*

- For object oriented languages: Explain Encapsulation, Inheritance and Polymorphism, provide examples. *Note: This is for positions that list object oriented programming as a requirement. If you do not know what these three terms, are a google search of object oriented and then each of the three words will provide a simple answer and probably a good wikipedia page. The answers given by a knowledgable developer will often be almost verbatim with any definition you find on the Internet.*

- When would you use global variables, if ever? *Note: This is a good question for seeing how they think. This might even cause the candidate to do some thinking on the fly to consider what a good answer would be. There is no right answer, but there are some that are essentially "wrong." The wrong answers include: all the time, never, and "what is a global variable?"*

- What type of development do you prefer? (see above table for examples) *Note: No right answers on this one, just a way to find out what they prefer and if they might be a good fit. This is also a question that might help highlight some complementary and/or crossover skills.*

- What type of team environment do you prefer? *Note: Make sure the answer is a match for the position(s) that are applied for. Always ask for some examples in their experience just to avoid weasels.*

- Explain applications you have written and the problem(s) they solved. *Note: Another "no wrong answer" question that will help provide a picture for the type of developer this is and their experience. Some developers will be fine until they get to the "problem it solved" part of this question. These are junior developers no matter what their resume says.*

Screening Tip:
We provide a large number of screening questions, but these are not always applicable. You should be able to match a screening question to a job requirement. If a screening question does not match back to a requirement… don't ask it.

- Explain the value of a debugger and alternative methods when a debugger is not available. *Note: This is a good question for rating experience and possibly thinking on the fly. The value of a debugger is going to be along the lines of being able to quickly look at code that is causing a problem and the state of the application right before, or when, the bug occurs. Alternative methods include application logging, console output, and message boxes or alerts.*

- What are some reasons for an application log? *Note: Debugging as mentioned before, but some advanced answers will include looking for performance degradation, activity peaks and valleys, and security audits.*

In general, it does not hurt to ask a few syntax questions, but the best insight will come from asking about specific experience and applications. Use the time spent screening to find out what a developer has done and how they did it. This will provide a far better measure of their experience than simply knowing how many years they spent with a title or working with a particular technology.

Screening Tip:
Ask about experience over technical details to get a better view into how the developer works and their likely ability to adapt to changes in the future.

Development Languages Resources

Resources for more information

If you haven’t had your fill, here are some more places to expand your knowledge:

http://www.transcender.com - A great third party source of certifications and training.

Lamguage/Framework	Web Site
Java	http://www.oracle.com/technetwork/java/index.html
Struts	https://struts.apache.org/
Hibernate	http://hibernate.org/
Spring	https://spring.io/
C#	https://msdn.microsoft.com/en-us/vstudio/hh341490.aspx
Python	https://www.python.org/
Django	https://www.djangoproject.com/
C/C++	https://isocpp.org/
PHP	https://secure.php.net/
Ruby	https://www.ruby-lang.org/en/
Visual Basic	https://msdn.microsoft.com/en-us/library/2x7h1hfk.aspx
Objective C	https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

Source Materials

https://en.wikipedia.org/wiki/Third-generation_programming_language
https://en.wikipedia.org/wiki/Second-generation_programming_language
https://en.wikipedia.org/wiki/Fourth-generation_programming_language - Wiki pages on the generations of programming languages.

https://en.wikipedia.org/wiki/Smalltalk - Smalltalk related dates

https://en.wikipedia.org/wiki/Ada_(programming_language) - Ada related dates

https://www.cs.utexas.edu/~scottm/cs307/javacode/codeSamples/SimpleWordCounter.java - Sample java code
Quick Reference Document Links:

Java - http://www.payscale.com/research/US/Job=Java_Developer/Salary
C# - http://www.payscale.com/research/US/Job=C%2523_Developer/Salary
C++ - http://www.payscale.com/research/US/Job=C%2b%2b_Developer/Salary
PHP - http://www.payscale.com/research/US/Job=PHP_Developer/Salary
Ruby - http://www.payscale.com/research/US/Job=Ruby_Software_Developer_
%2f_Programmer/Salary